Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

ÁREA DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TÍTULO DEL TRABAJO

D. SERRANO CAMBLOR, Daniel
TUTOR: D. Juan Carlos Alvarez Alvarez

FECHA: Septiembre de 2020

Índice

Memoria

1. Resumen.

El presente trabajo pretende estudiar el efecto que produce la dinámica del robot y el algoritmo de navegación del mismo mediante el uso de ROS2, que nos facilitará tanto la visualización de los distintos procesos desarrollados por el robot como la conexión con programas de simulación y visualización que nos permitirán poder medir los parámetros que sean requeridos estudiar en las simulaciones. Para llegar a los objetivos usaremos un robot tipo Burger, que será explicado con detalle más adelante, y un algoritmo de navegación teniendo en cuenta la dinámica del movimiento.

Palabras clave: Inteligencia Artificial, ROS2, burger, C++, Python, nodos, robot

2. Abstract.

The present work aims to study the effect produced by the dynamics of the robot and its navigation algorithm through the use of ROS2, which will facilitate both the visualization of the different processes developed by the robot and the connection with simulation and visualization programs that they will allow to measure the parameters that are required to study in the simulations. To reach the objectives we will use a Burger-type robot, which will be explained in detail later, and a navigation algorithm keeping in mind the dynamics of movement.

Keywords: Artificial Intelligence, ROS2, burger, C ++, Python, nodes, robot

3. Objetivos.

El objetivo del trabajo será comprobar el efecto de añadir las características dinámicas propias del mundo real a la simulación, desde rozamiento que se produce por el contacto entre las ruedas y el suelo, hasta las inercias propias del movimiento del robot. Además, estudiaremos el efecto que produce las alteraciones en el algoritmo de navegación con el que trabajaremos y probaremos los efectos con diferentes simulaciones al modificar parámetros internos del código de navegación.

El robot que usaremos para el problema será el conocido robot tipo Burger, que será estudiado con detalle más adelante. Se trata de un robot bien conocido en el mundo, especialmente en el ámbito de la investigación. En él vamos a cargar un algoritmo de navegación que estudiaremos a fondo hasta ser capaces de modificar parámetros internos con diferentes objetivos. Comprobaremos además el efecto de los cambiar la dinámica interna del robot y cómo le afectará.

Buscaremos alcanzar un sistema capaz de operar en entornos cuyos terrenos no sean lineales, con dinámicas de movimiento, trabajando con el software ROS2, que se explicará en apartados posteriores, que nos permitirá programar las acciones que pueda llevar a cabo el robot y los entornos de simulación en los que busquemos probar los programas realizados. Como objetivos inherentes a los objetivos principales podemos mencionar los siguientes.

- Conocer y explorar ROS2: Framework innovador que facilitará el diseño y comprensión de acciones con robots, de tal forma que se aprenderá a instalar las distribuciones compatibles con nuestro objetivo, las arquitecturas del espacio de trabajo y las diferentes opciones de programación compatibles.

- Ubuntu como sistema operativo: Aprenderemos a usar, a nivel usuario, las principales funcionalidades de Linux en Ubuntu debido a que ROS2 funcionará correctamente en este sistema operativo.

- Familiarización con Turtlebot 3: Se trata de un paquete para ROS2 que nos facilita tres tipos de robot y varias capacidades de funcionamiento. Tendremos que escoger el robot que mejor se adapte a nuestras necesidades de estudio. Hay al menos tres tipos

de robot Turtlebot3 con distintas capacidades y características, nosotros trabajaremos con el robot tipo burger.

- Diseño de simulaciones: Dada la situación actual, se probarán en entornos diseñados virtualmente mediante el uso del programa Gazebo. Con lo cual tendremos que aprender a usar gazebo y a programar los escenarios en los que probaremos nuestro robot y sus capacidades.

- Recoger datos de las simulaciones que serán tratados con hojas de texto. Dichos datos podrán ser de diferente índole en función de lo que estemos buscando analizar.

Normalmente un robot trabaja en un entorno controlado con unas características para su movimiento de "relativa" simplicidad, como podría ser un robot trabajando en una fábrica donde el suelo es plano y su forma de llevar a cabo una acción suele ser de forma repetitiva. Si el robot trabajara en un entorno que no está controlado veremos qué factores habrá que tener en cuenta a la hora de implementar la programación del algoritmo de navegación.

4. Estado del arte.

4.1.- Breve historia de la robótica.

El término robot es acuñado a raíz de la palabra checa robota, acuñada por el escritor checo Karel Čapek en su obra *R.U.R. (Robots Universales Rossum)*[1] escrita en 1920, que significa trabajos forzados o servidumbre. Pero esto no significa que el ser humano no hubiese trabajado antes con "robots", entendiéndolos en este caso como autómatas.

Una de las primeras descripciones de autómatas aparece en la antigua China en el texto Lie Zi, atribuido al autor Lie Yukou[2], en el que se describe el encuentro entre el rey Mu de Zhou y un "artifice" llamado Yan Shi en el cual Shi presenta al rey una obra mecánica humanoide de tamaño natural.

En la antigua Grecia se describen cientos de máquinas y autómatas, pero cabe destacar el conocido invento Eolípila de Herón de Alejandría[3]. La máquina consistía en una cámara de aire con tubos curvos por los cuales el vapor expulsado generaba una fuerza que hacía girar el mecanismo. Autómata proviene de la palabra griega αὐτόματος que significa con movimiento propio que ya salía en la obra de Homero[4][5].

Figura 4.1: Reconstrucción de Eolípila.

Durante el siglo de oro del islam, en el siglo XIII durante la Edad Media, el inventor Al-Jazarí desarrolló varios proyectos con autómatas, desde autómatas musicales hasta autómatas para el lavado de manos[6].

Figura 4.2: Fuente con forma de pavo real diseñada por Al-Jazarí.

Dos siglos después, en el año 1495, el famoso polímita florentino Leonardo da Vinci diseñó un autómata humanoide que fue construido siguiendo los bocetos descubiertos en 1950. El autómata es un guerrero vestido con una armadura medieval que es capaz de hacer movimientos como sentarse, mover los brazos, el cuello o la mandíbula[7].

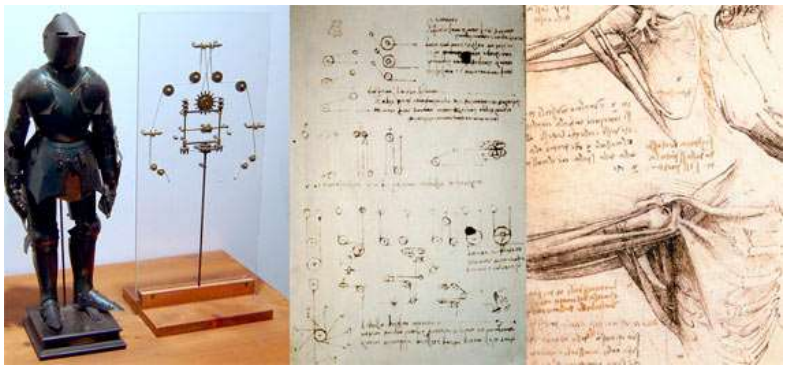

Figura 4.3: En la fotografía de la derecha vemos la reconstrucción del robot de Leonardo, en el centro sus anotaciones y a la derecha apuntes de articulaciones humanas.

En 1738 Jacques de Vaucanson, ingeniero e inventor francés, dedicó su vida a inventar distintos tipos de autómatas con el objetivo de despertar la fascinación del público. De entre todas las invenciones autómatas cabe destacar el conocido como Canard digérateur, que consistía en un pato mecánico capaz de comer, agitar las alas y excretar[8].

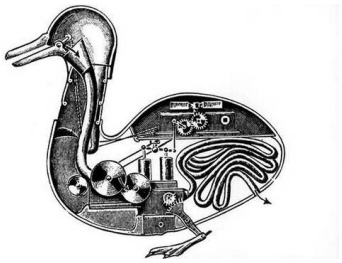

Figura 4.1: Diseño del canard digérateur.

Durante los siglos XVIII y XIX en Japón se usaban autómatas de madera conocidos como Karakuri que se traduce como “aparatos mecánicos para producir sorpresa en una persona”. Principalmente se usaban para transmitir leyendas y mitos tradicionales, aunque también se utilizaban para servir el té o incluso lanzar flechas con el arco[9].

Figura 4.2: Muñecos karakuri.

En 1930 Joseph Barnett, ingeniero de Westinghouse Electric Corporation, inventa lo que se considera el primer robot humanoide de la historia conocido como Elektro. Elektro era capaz de moverse, hablar o incluso fumar[10][11].

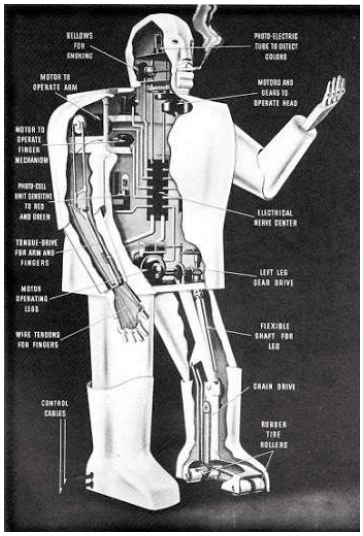

Figura 4.3: Robot Elektro.

En 1942 el famoso escritor Isaac Asimov publica “Círculo vicioso” donde incluye las conocidas tres leyes de la robótica[12].

Una década después, George Devol[13] y Joseph Engelberger[14] fundan la primera compañía de robótica industrial, Unimation, y crean en 1956 el primer robot industrial basandose en una patente de Devol. En 1961 la compañía instala el primer robot industrial conocido como Unimate en una cadena de montaje de la conocida compañía General Motors. El robot transportaba piezas fundidas en molde hasta la cadena de montaje y soldaba estas partes sobre el chasis del vehículo[15].

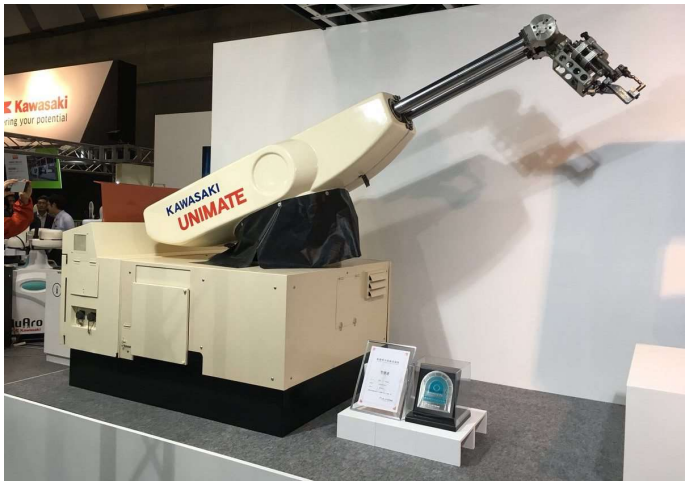

Figura 4.4: Robot unimate.

Durante la Guerra fría la Unión Soviética consigue aterrizar exitosamente en la superficie de marte, aunque pocos segundos después pierden el contacto con él, el primer robot soviético conocido como PROP-M en 1971[16][17].

En 1973 la conocida empresa KUKA Robot Group lanza al mercado el primer robot con seis ejes accionados por motor eléctrico[18].

Unos pocos años después Victor Scheinman, mientras trabajaba en la ya citada empresa Unimation, desarrolla el conocido robot PUMA en 1975. Consiste en un brazo manipulador programable universal[19].

Tres años después, en 1976, la NASA consigue poner el primer robot estadounidense, conocido como Viking I, en la órbita marciana desde la cual es capaz de enviar imágenes y el 20 de julio de ese mismo año aterrizó el robot en Marte[20].

Con los posteriores avances científico-tecnologicos, los robots han evolucionado hasta hoy día donde podemos convivir a diario con ellos estando muy presentes en la vida cotidiana, en la industrial y en las investigaciones científicas.

Actualmente, se agrupan comúnmente los grupos de robots dependiendo de sus características y su cronología.

- 1ª Generación: Se tratan de los llamados robots manipuladores. Se trata de robots que cumplen una secuencia de funciones “sencillas” preprogramadas.

- 2ª Generación: En esta generación se engloban los robots de aprendizaje. Estos robots siguen los movimientos de un operador humano y los memorizan.

- 3ª Generación: En esta generación se conocen como robots con control sensorizado. En esta generación los robots la computadora ejecuta un programa cargado y envía las ordenes al manipulador que a su vez gracias a la capacidad que tienen de recibir información de los sensores es capaz de decidir la acción a ejecutar. En definitiva, un lazo cerrado de control.

- 4ª Generación: En la nueva generación los robots son conocidos como robots inteligentes. Los robots inteligentes se caracterizan por sensores sofisticados que envían información al controlador que analizan y ejecutan ordenes de control complejas.

5.- Herramientas de software.

5.1.- ROS.

ROS son las siglas en inglés de Robot Operating System. Es un entorno de trabajo enfocado en el desarrollo de software robóticos que proporciona la funcionalidad de un sistema operativo que facilitará un clúster heterogéneo.

ROS nace en el laboratorio de Inteligencia Artificial de Stanford para dar soporte al robot STAIR de Stanford en 2007. Eric Berger y Keenan Wyrobek, estudiantes de doctorado, desarrollaron un sistema base para la robótica que proporcionara estructuras básicas para facilitar el trabajo y comprensión de la robótica. Con el objetivo de que fuese accesible para todo el mundo y que cada usuario pudiese configurar e interactuar con las herramientas y bibliotecas desarrolladas, ROS se concibió como código abierto.

Uno de los grandes atractivos de ROS para las corporaciones interesadas en el mundo de la robótica es que el ecosistema ROS permite que cualquier grupo pueda iniciar su propio repositorio de código ROS en sus propios servidores manteniendo la propiedad y control de este. Además, si la corporación estuviese interesada, pueden publicar el repositorio en plataformas públicas para permitir que usuarios de todo el mundo testeen el contenido y puedan reportar posibles errores o incluso implementar mejoras de todo tipo. [21]

5.1.2.- Comprensión de su arquitectura básica.

Los procesos se representan como nodos en una estructura gráfica en la cual los nodos se encuentran conectados entre los conocidos como topics. Los nodos pueden enviar mensajees entre sí a través de los topics ya sea enviando o recibiendo los mensajes habiendo sido suscritos previamente a los topics. Todo este proceso vendrá gestionado por el nodo maestro, denominado ROS MASTER, que se encargará de registrar y gestionar las suscripciones por parte de los nodos a los topics.

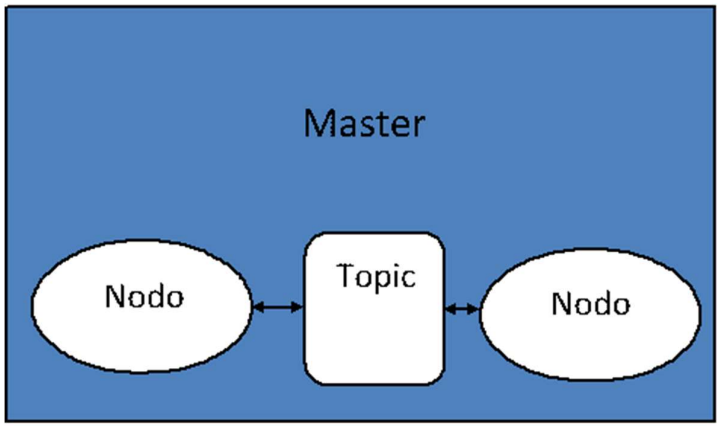

Ilustración 5: Gráfico de la arquitectura ROS. Los nodos intercambian mensajes a través del topic mientras el Master engloba los nodos y topics.

ROS se organiza mediante paquetes y pilas. Los primeros son conjuntos de nodos que llevan a cabo funcionalidades concretas como por ejemplo el movimiento de un robot. Los segundos en cambio, son conjuntos de nodos que unidos forman funciones complejas como por ejemplo un robot en movimiento capaz de medir la información de los objetos que tiene a su alrededor para esquivar distintos obstáculos.

El soporte de ROS de forma nativa es en C++ y Python pero se pueden implementar códigos en Java y otros lenguajes de programación mediante softwares de terceros que se adaptan a ROS.

5.2.- ROS2.

Debido a los cambios en el mundo de la robótica ROS fue actualizándose cada poco lanzando a lo largo de los años más de diez distribuciones distintas que buscaban mejorar las distribuciones anteriores y manteniéndose algunas en funcionamiento a día de hoy. Pese a lo anteriormente descrito se busca siempre mejorar aquello que se ha hecho con lo cual se lanza en 2017, y tras varios años de testeo con varias Alpha y Beta de prueba, la primera distribución de ROS2 que es bautizada con el nombre Ardent Apalone. Con ROS2 se mantienen aquellos aspectos positivos de ROS y se mejoran aquellos que quedaban obsoletos o que dificultaban el trabajo con ROS.

Los objetivos por los que se desarrolló ROS2 se deben a un crecimiento de la comunidad y de las funcionalidades de ROS en comparación con el uso que se le estaba dando. Dada la capacidad de

ROS y la importancia que obtuvo más allá del ámbito de la investigación, se decidió desarrollar ROS2 buscando alcanzar varias mejoras con respecto a ROS.

- Equipos de múltiples robots: En ROS se podía construir sistemas de múltiples robots, pero sin un enfoque estándar que se basaba en crear un "maestro" de ROS que gestionara todo.

- Pequeñas plataformas integrada: Facilitar que microcontroladores "bare-metal" y pequeñas computadoras puedan participar del entorno que proporciona ROS.

- Sistemas en tiempo real: Incluir la capacidad de control en tiempo real en ROS tanto en comunicación entre procesos como entre máquinas.

- Redes no ideales: Mejorar la conectividad de ROS cuando las conexiones online sufren perdidas y/o retrasos.

- Entornos de producción: Adecuar la capacidad del uso de ROS más allá de los laboratorios de tal forma que aquellos proyectos desarrollados en investigación puedan ser implementados con ROS fácilmente al mundo real.

- Patrones prescritos para la construcción y estructuración de sistemas: Proporcionar patrones claros y herramientas de soporte manteniendo la flexibilidad de ROS.

Uno de los puntos importantes en ROS2 es la facilidad para ordenar llamar a los nodos distintos y topics de tal modo que podemos trabajar fácilmente con las simulaciones, registrar los datos que nos interesen, como por ejemplo registrar las velocidades del robot, y visualizar la interactuación entre los distintos nodos y los topics que conectan unos con otros.

El núcleo de ROS es ser un sistema middleware de publicación y subscripción anónimo creado desde cero habiendo creado sus propios sistemas de descubrimiento, definición de mensajes, serialización y transporte, que fueron siendo mejorados a lo largo de los años.

En cambio, en ROS2 el gráfico ROS es el corazón del sistema. El gráfico ROS se refiere a los nodos en un sistema y a las conexiones entre ellos mediante las cuales se comunican. En ROS2 se

mantienen las bibliotecas de cliente rclcpp (biblioteca de cliente de c++) y rclcpy (biblioteca de cliente de Python) y además añade el uso de bibliotecas de cliente de ROS2 desarrolladas por la comunidad.

Los nodos son automáticamente detectados por el middleware de ROS2 siguiendo la siguiente secuencia de acontecimientos:

1. Cuando se inicia un nodo, anuncia su presencia a otros nodos en la red con el mismo dominio. Los nodos responden con información de ellos mismos para que se puedan realizar las conexiones adecuadas y los nodos se comuniquen.
2. Los nodos anuncian periódicamente su presencia para que si entrase en juego nuevas entidades, puedan realizarse las conexiones necesarias entre los nodos y las nuevas conexiones.

3. Los nodos al desconectarse informan a otros nodos.

ROS2 permite que cada complemento exista como su propio nodo en el gráfico ROS, con sus propios parámetros. Los complementos tampoco tienen que preocuparse de las colas de evoluciones de llamadas porque ROS2 maneja todo ello de forma interna. [22]

5.2.1.- Organización del espacio de trabajo y programación en ROS2.

El espacio de trabajo es vital si queremos implementar nuestras aplicaciones propias. Para poder tener un espacio de trabajo operativo donde ROS2 sea capaz de cargar los paquetes que deseemos y generar la red de conexiones de nodos y topics necesarias para nuestro objetivo, necesitamos un orden concreto en la carpeta de trabajo.

Los paquetes en ROS son contenedores de código y en ellos debemos encontrar los ficheros que definen el paquete y cómo debe ejecutarse.

Si trabajamos paquetes en C++ debemos tener dos archivos para definir el espacio de trabajo.

- **package.xml**: Se trata de un archivo en el que se contiene metainformación sobre el paquete
- **CMakeLists.txt**: se trata de un archivo en el que se describe cómo construir el código que está almacenado en el paquete

Un espacio de trabajo puede contener tantos paquetes como consideremos necesarios para el objetivo que busquemos. En nuestro caso el espacio de trabajo contiene varios paquetes que definirán el comportamiento de nuestras simulaciones y la operación del robot.

```
espacio_de_trabajo/
   src/
     my_package/
         my_package/
           CMakeLists.txt
           package.xml
         my_package_teleop/
           setup.py
           package.xml
           setup.cfg
           resource/
           my_package_teleop/
             __pycache__/
             script/
               __pycache__/
               __init__.py/
               teleop_keyboard.py
             __init__.py
     my_package_simulation/
         include/
           my_package_simulation/
             Robot_drive.hpp
         launch/
         models/
         worlds/
         rviz/
         src/
```

```
 Robot_drive.cpp
CMakeLists.txt
package.xml
```

La carpeta my_package contiene los nodos que cargaremos cuando lancemos las simulaciones que están almacenadas en my_package_simulation.

En la carpeta Launch almacenamos los archivos ejecutables para cargar desde la línea de comandos de Ubuntu. En ROS2 los archivos ejecutables launch han cambiado suestructura y ahora tan solo hay que escribirlos terminando con extensión .launch.py.

En la carpeta models almacenamos los archivos que definen el Robot con el que vayamos a trabajar.

La carpeta worlds están almacenados los entornos de simulación que desemos con las características que busquemos para que opere el robot.

La carpeta rviz contiene un archivo .rviz que sirve para cargar el programa Rviz2. En nuestro caso, Rviz2 nos interesa porque es donde cargaremos el mapa de navegación con el que podremos interactuar enviando la posición que buscamos inicial y final del robot en dicho mapa.

Una vez hayamos terminado de crear los paquetes y los códigos que necesitamos, tenemos que "construir" todo lo que hemos almacenado en nuestro espacio de trabajo. Ubuntu proporciona una línea de comando por teclado para construir los espacios de trabajo que deseemos.

```
colcon build
```

Una vez hecho, tendremos en la capa superior de nuestro espacio de trabajo carpetas generadas tras la construcción del espacio de trabajo.

```
install build log src
```

La programación de los distintos nodos usados al igual que los entornos de simulación en Gazebo se han diseñado con Python y C++ puesto que son los lenguajes de programación que mejor conozco y gracias a las facilidades de lectura entrecruzada que presenta, de base, ROS2. ROS2 facilita la lectura de distintos repositorios que tengan diferentes lenguajes de programación por lo que hace muy simple variar entre uno u otro dependiendo de aquello que necesitemos cubrir.

5.3.- Simulador.

Debido a la necesidad que tenemos como ingenieros de probar aquello que hemos diseñado para corroborar que funcione correctamente y que no haya fallos, vamos a tener que probar todo lo que hayamos programado en un entorno de simulación controlado que sea compatible con el espacio de trabajo en el que estamos, ROS2, y que sea capaz de proporcionarnos las mayores facilidades posibles para generar los entornos que deseamos crear.

Hay muchos entornos de simulación que son compatibles con ROS y otros tantos que pueden ponerse en funcionamiento si usamos un sistema operativo u otro. A continuación, veremos algunos de los principales entornos de simulación.

- Gazebo

  Gazebo es un entorno de simulación potente donde se puede diseñar robots a nuestro gusto, cambiar las físicas del entorno, añadir mundos prediseñados por la empresa o crear los nuestros propios. Además, tiene un amplio espectro de robots que pueden ser usados con libertad puesto que el programa es completamente gratuito. Trabajar con Gazebo es fácil e intuitivo y la propia empresa provee en su página una gran cantidad de tutoriales que facilitan comprender cada uno de sus aspectos.

  Es un entorno de simulación muy potente y eso consume muchos recursos de la computadora en la que estemos trabajando con lo cual es importante saber qué capacidad tiene nuestro ordenador para trabajar de forma fluida con nuestras simulaciones.

- Morse

Es un simulador académico que está orientado a la simulación 3D em entornos grandes.

El simulador deja que el programador diseñe por sí mismo las dinámicas y cinemáticas con las que se vaya a trabajar. Utiliza sctipts de Python para programar cada aspecto que deseemos simular.

- V-rep

  Es un simulador potente en el que se disponen de dos posibles licencias, educacional o comercial. Permite dotar de físicas personalizadas a nuestro entorno. El software proporciona una interface integrada que permite diseñar todo el entorno y añadir con facilidad. Se puede trabajar en varios lenguajes de programación, aunque se trabaja principalmente con C++.

En nuestro proyecto escogemos trabajar con Gazebo, versión 9+, puesto que es un entorno de simulación muy potente que fue concebido para probar rápidamente los diseños del programador. Además, proporciona un amplio espectro de librerías con diferentes objetos, robots y mundos con los que podemos interactuar y añadir a nuestras escenas para probar nuestro proyecto.

5.3.1.- Gazebo.

Gazebo es un potente entorno de simulación que está especializado en robótica y simulaciones con ROS. Es capaz de definir parámetros de simulación teniendo en cuenta diferentes aspectos de la dinámica de los objetos, ya sea desde poder cambiar la fuerza de la gravedad o el efecto del viento hasta cambiar las dinámicas de los objetos, pudiendo alterar el peso, la rugosidad, el rozamiento, etc…

Con Gazebo podemos crear robots tal y como deseemos haciendo que cada parte del robot tenga sus características particulares pudiendo definir las uniones entre los distintas partes del robot con sus características particulares con el objetivo de hacer las simulaciones adecuarse a nuestras necesidades.

La forma de definir las características o partes del robot, así como el entorno de simulación en concreto, se hacen siguiendo un formato de descripción de simulación especializado diseñado por Gazebo llamado SDFormat.

Una característica interesante de Gazebo es que podemos recopilar toda la información que deseemos de la simulación. Podemos ver a tiempo real los efectos de la simulación en los parámetros que deseemos estudiar, de tal forma que si queremos ver cómo evoluciona la velocidad de movimiento en un robot podemos comprobarlo creando una gráfica en la cual se irá actualizando a tiempo real los valores que toma a lo largo del tiempo la velocidad del robot de estudio. Además, una vez hemos finalizado nuestra simulación podemos guardar la información de la gráfica generada como un PDF, en caso de que nos interese tan solo ver la forma de la gráfica, o como un archivo CSV, en caso de que queramos exportar la información a una hoja de cálculo para realizar un estudio preciso de los valores de la simulación.

Desde la implementación de ROS2 Gazebo ha trabajado para mantenerse como una de las plataformas de simulación que mejor se complementa con ROS2. Por ello han aprovechado las ventajas que presentaba ROS2 frente a ROS, como el trabajar prescindiendo del Maestro o reducir el código interno. Por ello Gazebo provee de un paquete exclusivo para ROS2 con complementos más pequeños y específicos

ROS2 consigue que los complementos existan como sus propios nodos en el gráfico ROS con sus parámetros y características propias. Los complementos no necesitan preocuparse de mantener las colas de devoluciones de llamadas. Además, ROS2 en Gazebo puede iniciarse sin ningún complemento y los complementos habilitados para ROS2 se pueden agregar en tiempo de ejecución.

5.3.2.- SDF.

El formato de descripción de robots en ROS es URDF (Universal Robotics Description Format) que es un formato de archivo XML (Extensive Markup Languaje). Son un formato útil y estandarizado, pero carece de muchas características de tal forma que solo puede definir las características cinemáticas y dinámicas de un robot de forma aislada. Además, en este formato no se puede definir coeficientes de fricción o deslizamiento ni tampoco podemos definir las características del entorno de trabajo, solo puede definir al robot.

SDFormat es un formato de descripción de simulación. Se trata de un formato XML que describe objetos y entornos para simulación, visualización y control de robots. Fue diseñado para aplicaciones científicas, pero con el tiempo se ha convertido en un formato estable capaz de describir todos los aspectos necesarios para crear un entorno de simulación realista.

Nuestros robots y las escenas de simulación son diseñadas con SDF y los archivos que creamos se guardan con extensión .sdf y un archivo .config asociado.

Para describir un robot en URDF o SDF tenemos que describir las articulaciones y los enlaces.

- Joints (Articulaciones): La función de las articulaciones es conectar dos enlaces (link), el enlace hijo (child link) y el enlace padre (parent link). Las articulaciones van a ser las que generan el movimiento en el robot y pueden ser movimientos descritos como prismáticos (prismathic), continuo (continous), de revolución(revolution) o fijo(fixed). En cada articulación debemos definir su posición y orientación de origen y fin que son posiciones relativas al child link y al parent link respectivamente. Por tanto, las articulaciones describen totalmente la cinemática del robot.

- Links (enlaces): Los enlaces definen propiedades como la masa o la longitud entre articulaciones. Tendremos que definir en cada enlace su peso, su posición de origen y fin tal que definamos su longitud, su posición y orientación y la matriz de inercia que define el comportamiento rotacional del solido rígido.

6.- Dinámica de movimiento.

El análisis que hacemos en este trabajo se basa en estudiar cómo afecta la dinámica real de un robot y del entorno de trabajo mediante simulaciones. Para ello vamos a explicar los principales parámetros dinámicos que vamos a estudiar.

6.1.- Masa inercial.

La inercia es la propiedad que tiene un objeto para oponerse a un cambio en el estado de movimiento en el que se encuentra. Por lo tanto, un cuerpo mantendrá su estado de reposo o movimiento relativo siempre que no se ejerza ninguna fuerza que pueda cambiar su estado. La inercia tiene por tanto una relación directa con las características físicas del objeto, y más especialmente con la masa.

La masa inercial es la resistencia de un cuerpo al cambio de velocidad. Matemáticamente se entiende por masa inercial a la comparación de las aceleraciones ante una misma fuerza.

$$F_1 = F_2 \rightarrow m_1 a_1 = m_2 a_2 \rightarrow m_2 = m_1 \frac{a_1}{a_2} \tag{5.1}$$

Por tanto, la masa inercial aumenta cuando la velocidad aumenta. Además, la masa del objeto afectará a inercia que define el objeto.[23][24]

6.2.- Momento inercial.

El momento inercial es una medida de la inercia rotacional de un cuerpo cuya representación es vectorial y representa la distribución de un cuerpo respecto a un eje de giro. Es análogo a la masa inercial en movimiento rectilíneo y uniforme. Es el valor escalar del momento angular longitudinal de un sólido rígido.

El momento de inercia se puede representar matemáticamente como el sumatorio de los productos escalares de las masas de las partículas en movimiento por el cuadro de la mínima distancia de cada partícula con el eje de rotación.

$$I = \sum m_i r_i^2 \tag{5.2.1}$$

Se puede generalizar en cuerpos de masa continua (nuestro caso) como:

$$I = \int r^2 dm = \int_V \rho r^2 dV \tag{5.2.1}$$

El subíndice de la integral determina que la integral se aplica sobre el volumen del cuerpo de estudio de tal forma que se trata de una integral triple.

En el caso de estudio que nos ocupa el cuerpo, trabajamos con un sólido en 3D y por ello se definirá la inercia como el tensor de inercia. El tensor de inercia de un sólido rígido es un tensor simétrico de segundo orden que viene dado por una matriz simétrica. La matriz es el tensor de inercia expresado en la base XYZ. La matriz vendrá entonces definida por los momentos de inercia, en la diagonal, y los productos de inercia que están en el resto de la matriz.[25][26]

$$Tensor\ de\ inercia = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{pmatrix} \tag{5.2.3}$$

6.3.- Fricción.

Debido a que buscamos visualizar los efectos de la dinámica en nuestras simulaciones, debemos conocer las ecuaciones matemáticas que definen el comportamiento de la fricción o rozamiento.

La fricción o fuerza de rozamiento es la fuerza que se ejerce entre dos cuerpos con superficies ásperas al entrar en contacto, que se opone al deslizamiento. Se genera debido a las imperfecciones superficiales de los objetos y generan un vector de fuerza que formará un ángulo con la fuerza normal.

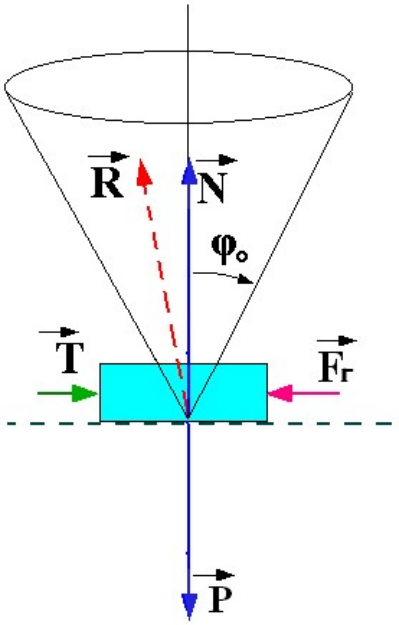

Ilustración 6: Pirámide de fricción. Fuerzas presentes en el movimiento de un objeto con rozamiento entre objeto y suelo.

Como podemos ver al aplicar una fuerza a un objeto se deben vencer los efectos de varias fuerzas para que éste se pueda mover. La fuerza de rozamiento actúa en la dirección contraria a la dirección a la que apunta el vector de fuerza que estamos aplicando. Para que el objeto se mueva la fuerza aplicada debe superar el umbral mínimo de movimiento que limita la fuerza de rozamiento que se conoce como fricción estática y siempre resultará menor que el coeficiente de fricción que hay entre los dos objetos en contacto multiplicado por la fuerza normal, que es igual a la masa del objeto por la aceleración de la gravedad. Una vez se ha vencido la fricción estática el objeto se empezará a mover y a lo largo de su movimiento habrá una fuerza que se opondrá a su movimiento de forma continuada y es la llamada fricción dinámica que es considerada constante.

La fuerza de rozamiento tiene dirección paralela a la superficie de apoyo y el coeficiente de rozamiento depende de la naturaleza intrínseca a los dos cuerpos que se encuentran en contacto y del estado en que se encuentren cada una de las superficies.

La fricción torsional se calcula en función de la superficie que esté en contacto y el radio de la superficie.

$$T = \frac{3\pi}{16} * a * c * N \quad (5.3.1)$$

Donde T es el par debido a la fricción torsional, a es el radio de contacto, c es el coeficiente de fricción torsional y N la fuerza normal en el contacto.

El cálculo de a se hace como se verá a continuación:

$$a = \sqrt{R * d} \quad (5.3.2)$$

Siendo R el radio de la superficie en el punto de contacto y d la profundidad del contacto.

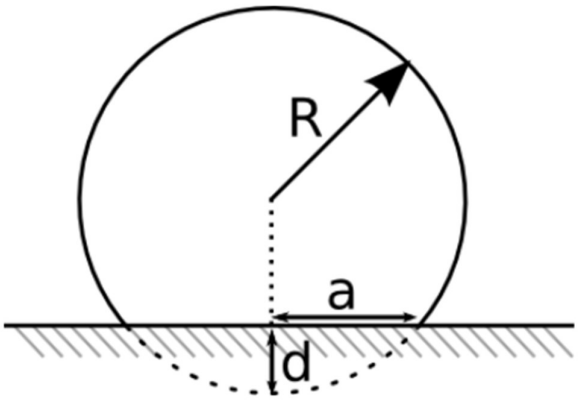

Ilustración 7: Variables para calcular el radio de contacto 'a'.

Los coeficientes de fricción de los materiales definen la oposición al deslizamiento en las superficies de dos cuerpos en contacto. En el caso que estudiamos el contacto se produce entre los neumáticos del robot y diferentes superficies en las que habrá mayor o menor fricción dependiendo de la superficie. El mundo de simulación diseñado tendrá distintas superficies y éstas variarán en cuanto a sus coeficientes de fricción asociados.[27][28][29]

6.4.- Deslizamiento.

El deslizamiento es una condición de baja resistencia al deslizamiento debido a una fricción insuficiente en la superficie de trabajo debido a una capa de fluido viscoso que está entre la superficie de apoyo y la superficie de trabajo. Debido a dicha capa, la adherencia entre el robot y la superficie

de trabajo se reduce drásticamente. Se trata de una forma de fricción donde se encuentra presente una capa de un fluido entre el sólido rígido y la superficie en seco. Cuanto mayor sea el valor de deslizamiento menor será la adherencia entre el robot y la superficie de trabajo.

La viscosidad se manifiesta en fluidos y gases. Se trata de la relación entre el esfuerzo cortante y el gradiente de velocidad. Si el índice de viscosidad es muy alto, el rozamiento entre las capas lo será también y por ello no podrían moverse unas con respecto a otras con lo que tendríamos un sólido. En cambio, si tenemos un índice de viscosidad muy bajo, el rozamiento lo será también, de tal forma que tendríamos un fluido.

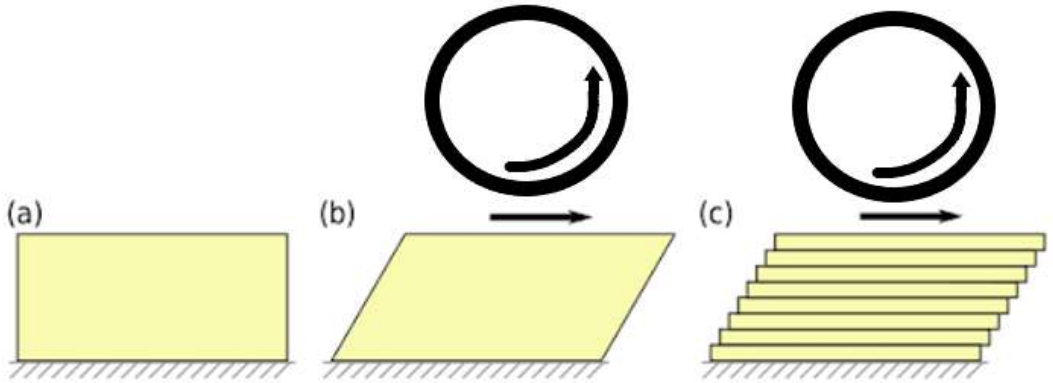

Ilustración 8: Deslizamiento por fluido. De izquierda a derecha: (a) representa el fluido en reposo sin efecto de fuerza tangencial superior alguna, (b) se ejerce una fuerza tangencial sobre el fluido que comienza a deformarse y (c) el desplazamiento relativo de las capas respecto de las adyacentes deforma el fluido debido a la fuerza tangencial que produce la rueda.

Un determinado par aplicado por un eje motor sobre una rueda permitiría desplazar una carga o por el contrario la rueda patinaría ocasionando una situación de deslizamiento sin rodadura (tal como sucede por ejemplo cuando un automóvil trata de arrancar sobre hielo o sobre un suelo en el que existe un fluido lubricante) si el valor del coeficiente de resistencia a la rodadura entre el radio de la rueda es menor o igual que el coeficiente de rozamiento.

$$\frac{\mu r}{R} \leq \mu \qquad (5.4.1)$$

Donde μ es el coeficiente de rozamiento, μr es el coeficiente de resistencia a la rodadura

6.5.- Terrenos no planos.

Terrenos no planos consideraremos a todos aquellos terrenos con desniveles de cualquier tipo y características propias que puedan influir en el desarrollo de una tarea al robot.

Las características, por ejemplo, de una elevación en el terreno afectarán o no al rendimiento del robot que vaya a tener que trabajar en dicho terreno en función de las limitaciones de hardware y de las limitaciones de software que tenga el robot.

Por ejemplo, un robot trabaja desplazándose desde un punto A a un punto B en un terreno lineal en condiciones ideales (alta fricción, sin deslizamiento y sin inclinaciones de terreno). Si añadimos una inclinación entre los dos puntos de 30º el rendimiento del robot se verá afectado de tal forma que tendrá que vencer fuerzas en este desplazamiento que en su forma original no estaban presentes.

En definitiva, los terrenos no planos son entornos comunes como puede ser ir caminando por la calle y encontrarse una calle inclinada. Con lo cual, resulta de interés investigar cómo afectan dichos terrenos al robot y a su software con el objetivo de mejorar y prever dichos problemas de la vida real.

7. Características del robot y del algoritmo de navegación.

7.1. Robot.

El paquete Turtlebot3 nos proporciona una variedad de 3 robots, Burger, waffle y waffle Pi, con los que podemos probarlos en entornos de simulación y recoger datos de los sensores integrados en ellos. El paquete contiene varios subpaquetes relacionados con funcionalidades que se les añade a los robots ya mencionados. Las funcionalidades principales son:

- Conexión con robot real.
- Operaciones básicas.
- SLAM.
- Navigation2.

El paquete Turtlebot3 es de libre acceso y completamente abierto a que se modifique internamente al gusto del usuario. Aún se encuentra en desarrollo, pero ya presenta una gran cantidad de opciones y aplicaciones de uso.

Nosotros vamos a trabajar principalmente con el paquete Navigation2 y la Teleoperación mediante el ordenador portátil.

7.1.1. Robot: Burger.

El robot que vamos a utilizar para hacer las simulaciones y extraer los datos que necesitamos será el tipo Burger.

El modelo Burger es un modelo muy común de robots de estudio, es versátil y se puede modificar su estructura mecánica a gusto y necesidad del usuario. En nuestro caso tan solo vamos a modificar variables internas del robot relacionadas con la dinámica de movimiento que lo define. El robot es de código abierto y es desarrollado y comercializado por ROS.

Se trata de un robot de dos ruedas cuyo movimiento se basa en la transmisión diferencial con lo cual el movimiento y giro se basará en la diferencia de movimiento angular de las ruedas a distintas velocidades.

El centro de masas, la inercia y los puntos donde habrá fricción podemos visualizarlos en Gazebo mediante una funcionalidad que está presente en el simulador.

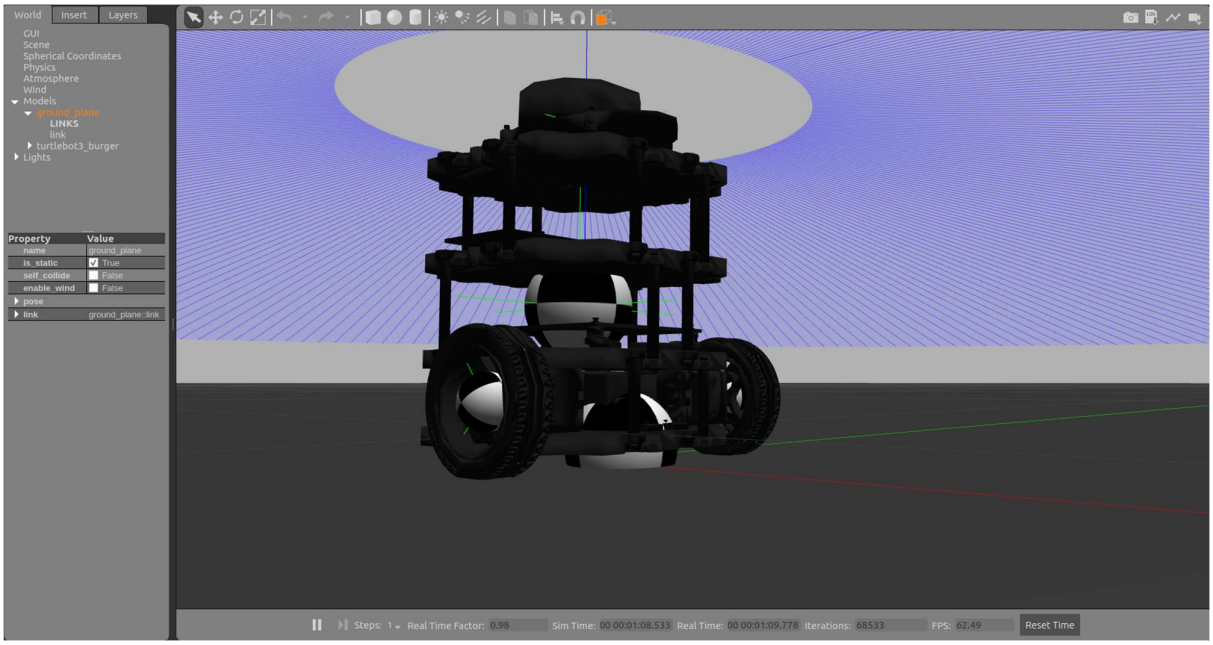

Figura 7.1: Centros de masas.

En la imagen superior podemos ver unas esferas de colores blanco y negro que representa el peso del robot en sus diferentes partes o uniones. Por ello podemos ver que las ruedas tienen su centro de masas y el cuerpo del robot tiene el centro de masas propio. Cuanto mayor peso tenga el robot mayor será el tamaño de la esfera, de esta forma podemos ver fácilmente y de forma visual el efecto de la masa del robot.

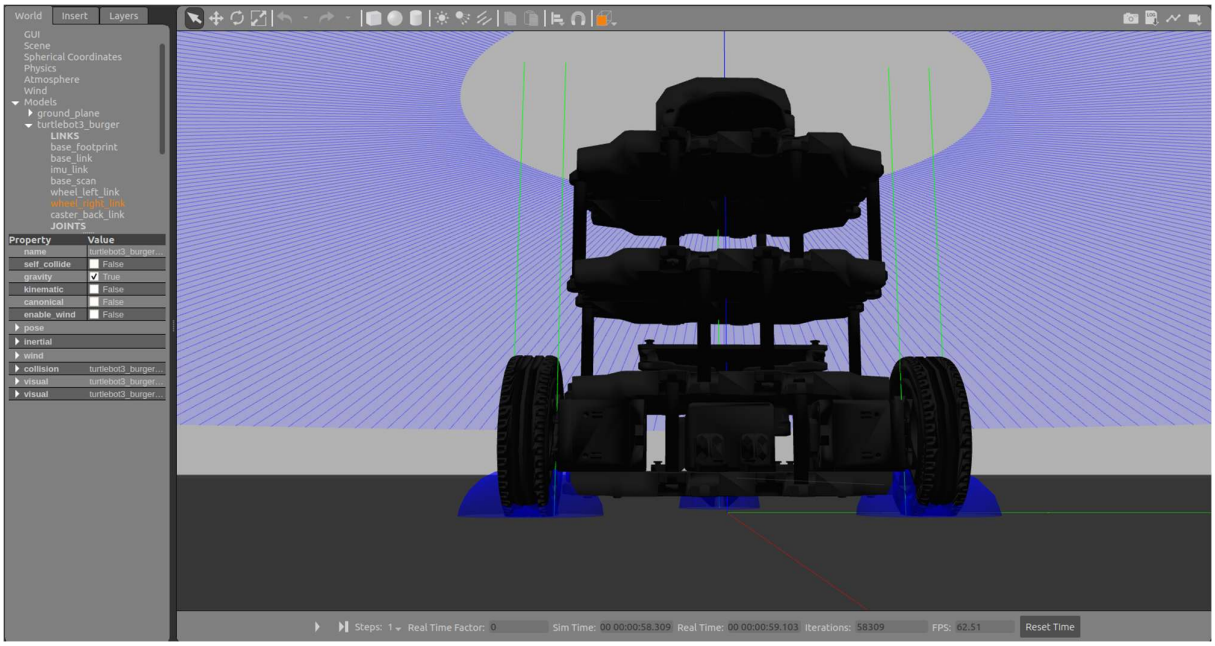

Figura 7.2: Puntos de contacto donde se producirá la fricción.

Como podemos observar en la imagen superior, hay tres esferas de color azul oscuro que representan los puntos de contacto entre la superficie por donde circulará el robot y el robot. En este caso, podemos ver tres esferas, dos se corresponden claramente al contacto entre las ruedas laterales del robot y el suelo, mientras que podemos ver una tercera esfera más alejada y de menor tamaño que se corresponde a una pequeña esfera en la parte posterior del robot que sirve para el equilibrio. En ellas se producirá el efecto de la fricción y del deslizamiento.

Por último, el movimiento del robot se produce por lo que se conoce como la conducción diferencial. Con ello hacemos referencia a que nuestro robot se moverá y podrá girar a un lado u otro dependiendo de la diferencia de velocidad de rotación de las ruedas laterales. Con lo cual no hay un eje que comunique directamente ambas ruedas, por lo que cada rueda podrá tener una velocidad de giro independiente a la otra rueda. Con ello, si queremos que el robot haga un desplazamiento rectilíneo, las ruedas girarán a una velocidad idéntica manteniendo la dirección deseada. Por lo contrario, si deseamos que el robot gire hacia un lado, la rueda más alejada del lado al que deseamos que gire el robot se moverá a mayor velocidad que la que está en el lado al que deseamos que se desplace el robot.

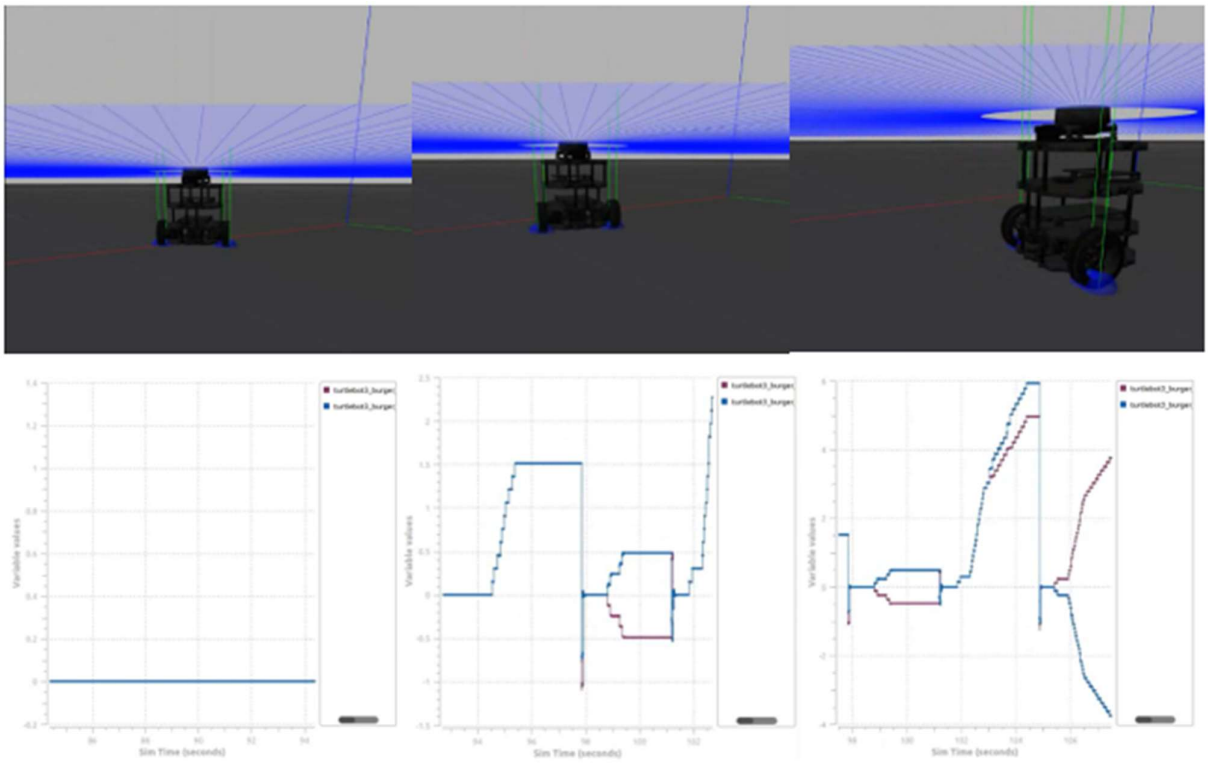

Figura 7.3: Movimiento diferencial. De izquierda a derecha: Robot y gráfica en instante de no movimiento, robot moviéndose sin giro y finalmente, el robot girando y moviéndose.

Podemos ver en las gráficas que al moverse el robot de forma rectilínea las dos ruedas giran a la misma velocidad, cuando el robot gira sobre sí mismo las ruedas giran a la misma velocidad, pero en sentido contrario y finalmente podemos ver como cuando el robot se está moviendo y a la vez está girando una de las ruedas (en la imagen sería la rueda derecha) gira a menor velocidad que la otra rueda de tal forma que gira mientras está desplazándose.

7.1.2. Robot: Burger especificaciones.

A continuación, veremos las especificaciones del fabricante para el turtlebot3 burger. Nos interesa especialmente las características que definen su física dinámica de movimiento, velocidades de traslación y rotación y peso del robot. [30]

Velocidad máxima de traslación	0,22 m / s
Velocidad máxima de rotación	2,84 rad / s (162,72 grados / s)

Carga útil máxima	15kg
Tamaño	138 x 178 x 192 mm
Peso	1 kg
SBC (computadoras de placa única)	Raspberry Pi 3 Modelo B y B +
MCU	ARM Cortex®-M7 de 32 bits con FPU (216 MHz, 462 DMIPS)
Solenoide	XL430-W250
LDS (Sensor de distancia laser)	Sensor de distancia láser 360 LDS-01
IMU	Giroscopio Acelerómetro de 3 ejes Magnetómetro de 3 ejes de 3 ejes
Conectores de potencia	3.3V / 800mA 5V / 4A 12V / 1A
Pines de expansión	GPIO 18 pines Arduino 32 pines
Periférico	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5 pines OLLO x4
Conexión a PC	USB
Adaptador de corriente	Entrada: 100-240 V, CA 50/60 Hz, 1,5 A @ máx. Salida: 12 V CC, 5 A

Tabla 7.1: Especificaciones del robot.

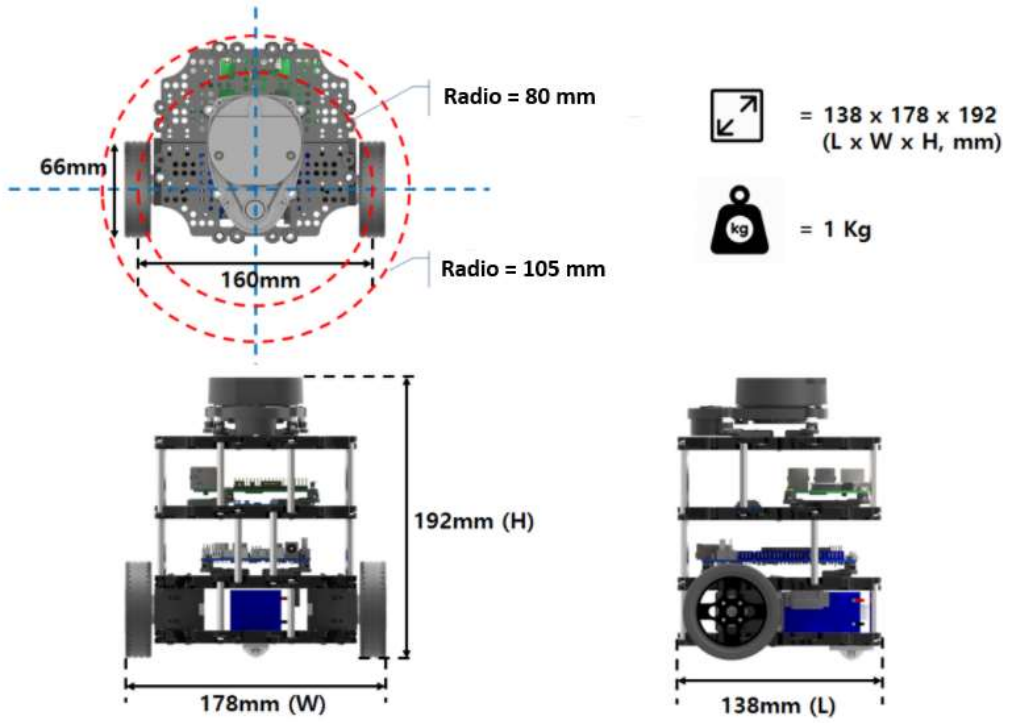

Figura 7.4: Robot burger plano con medidas y peso total.

7.2. Algoritmo de navegación.

Los algoritmos son un conjunto de operaciones que nos permiten hallar la solución a los problemas planteados. Dado que nosotros buscamos que el robot se mueva de un punto a otro lo más rápido posible esquivando obstáculos en el camino, necesitaríamos programar una serie de algoritmos, como puede ser calcular la distancia entre el robot y el entorno y decidir el camino correcto para evitar obstáculos y llegar al destino deseado, en ROS2. Hay una gran cantidad de librerías compartidas de uso libre donde los algoritmos que definen la navegación del robot ya están programados. En nuestro caso vamos a trabajar con Navigation2.

Se trata de un paquete con una gran cantidad de funcionalidades para la robótica, es libre acceso y facilita que los usuarios puedan manipular los parámetros de los distintos algoritmos del paquete para que se cubran mejor las necesidades de cada proyecto.[31]

7.2.1. Servidor de acción.

Los servidores de acción de ROS2 son la forma de controlar tareas de larga duración como es el caso de la Navegación. Un servidor de acción se basa en que un cliente solicita ejecutar una tarea, pero ésta dura mucho tiempo. Para poder completar una tarea de larga duración, los servidores de acciones y los clientes llaman a la tarea de larga duración en otro subproceso y devolver más adelante su resultado. Una vez tenemos en un subproceso la llamada de la tarea principal, podemos dejarla bloqueada hasta que se complete la acción. Para explicar estos conceptos a modo de ejemplo, imaginemos que queremos que se doble un brazo robot un ángulo deseado, para ello podemos decir que la solicitud sería el ángulo deseado, los mensajes que recibimos en el proceso serían el ángulo que nos queda por desplazar para llegar al ángulo deseado y el resultado sería una variable tipo booleano que nos dice si ha llegado al objetivo o no.

7.2.2. Servidores de navegación.

Los planificadores y controladores son el núcleo de una tarea de navegación. Los recuperadores se utilizan para ayudar al robot a salir de una situación inesperada o para ayudar a solucionar varios tipos de problemas para que el sistema sea tolerante a fallos concretos.

Los servidores del planificador y del controlador se configuran en tiempo de ejecución con los nombres, alias de la tarea que se ejecuta, y tipos, nombres de pluginlib que se han registrado, de algoritmos que se utilizarán. Cuando el árbol de comportamiento marca el nodo correspondiente, realiza la llamada al servidor de acciones para procesar su tarea.

Cada recuperación del servidor de recuperación, contiene su propio nombre y a su vez expone su propio servidor de acción debido a la alta variedad de acciones de recuperación que pueden crearse.

7.2.3. Planificadores.

El planificador se encarga de calcular la ruta para completar alguna función objetivo. El planificador para lograr el objetivo tendrá acceso a los datos compartidos por los sensores que almacenaran la información del entorno de trabajo y a la representación ambiental del entorno de trabajo.

La tarea general en Navigation2 para el planificador es calcular una ruta válida desde la pose inicial del robot hasta la pose objetivo, lo que conlleva que deberá desplazarse en el espacio de trabajo y además orientarse hacia la posición deseada.

7.2.4. Controladores.

Los controladores son la forma en que seguimos la ruta calculada globalmente o completamos una tarea local. El controlador tiene acceso a una representación del entorno local para intentar calcular los esfuerzos de control factibles para la base a seguir. Muchos controladores proyectarán al robot hacia adelante en el espacio y calcularán la ruta localmente factible en cada iteración de actualización.

El controlador en Navigation2 ejecutará como tarea principal calcular el esfuerzo necesario para mantener la ruta planificada de forma global.

7.2.5. Recuperaciones.

El objetivo de las recuperaciones es lidiar con condiciones desconocidas o de fallo del sistema y manejar estos fallos de manera autónoma. Los fallos más comunes que podemos solventar son, por ejemplo, detección de obstáculos inexistentes en el mapa de navegación que se puede deber a una desorientación del robot, o que el robot se atasque debido a obstáculos dinámicos o por un control deficiente.

7.2.6. Guía de configuración.

Como habíamos dicho anteriormente, el paquete Navigation2 puede ser modificado por el usuario que va a trabajar con él para cubrir sus necesidades a su antojo. Los siguientes son los paquetes con las posibles modificaciones que podemos realizar. [32]

- Mapa 2D.
    - Inflation_layer.inflation_radius: Área de inflado de obstáculo de tal forma que el camino se planificará para evitar atravesar el área.

- Inflation_layer.cost_scaling_factor: Factor por el cual cuanto más pequeño el número mayor lejanía tomará nuestro robot con respecto al obstáculo que se haya detectado.

- Controlador DWB.
  - Max_vel_x,max_vel_y, min_vel_x y min_vel_y: Valores máximo y mínimo de traslación del robot en los ejes de coordenadas x e y. Si establecemos un valor mínimo negativo el robot podrá ir marcha atrás. Los números son leídos en m/s.
  - Max_vel_theta y min_vel_theta máximos y mínimos de velocidad de rotación.
  - Valores de aceleración y desaceleración máximos en m/s^2 para la rotación y traslación del robot.
  - Xy_goal_tolerance: Valores de distancia y orientación permitidas al que el robot detecta haber llegado a la meta.
  - Transform_tolerance: Tiempo en segundos de tolerancia a la lectura de las posiciones y orientaciones de las partes del robot durante la simulación.
  - Sim_time: Factor que adelanta a la simulación en segundos. Un valor demasiado alto no permite girar de forma brusca.

8. Resultados experimentales.

Para estudiar los efectos explicados en apartados anteriores, hemos realizado varias simulaciones con los parámetros que queremos probar. La selección de los parámetros de estudio dependerá del objetivo de nuestras simulaciones. Por tanto, para probar los parámetros de estudio son la velocidad y la aceleración de los robots implicados en dicha simulación. Para comprobar el efecto de la masa del robot, hemos usado como parámetro de estudio la posición que ocupa en el espacio de simulación cada robot con el objetivo de probar si dependiendo de los diferentes valores de peso que tenga el robot influirá en su posición final. Y para comprobar el efecto de terrenos no lineales, hemos hecho dos simulaciones donde comprobaremos cómo afecta un terreno no lineal en la posición del robot, la velocidad y la aceleración.

8.1. Efecto de la fricción.

Para simular el efecto de la fricción que se produce entre rueda y suelo, vamos a modificar el entorno de simulación con suelos que tengan diferentes valores de fricción con respecto a la rueda. Usaremos cuatro robots, con los valores predeterminados, en diferentes pistas con distintos valores de fricción. Pediremos que los robots se muevan a una velocidad fija durante un tiempo limitado y mediremos la velocidad, la posición y la aceleración en cada uno de los robots para extraer los datos relevantes de la simulación.

Para poder realizar esto en Gazebo tenemos que cambiar los valores de las pistas con las que trabajaremos en cuanto al valor de fricción. Por tanto, vamos a tener tres pistas de circulación con valores distintos de fricción que para que sean leídos desde el formato SDF la variable se denomina mu.

En las tres simulaciones vamos a pedir por teclado una velocidad de 0.22 m/s y simularemos el tiempo suficiente para que los robots puedan alcanzar la velocidad pedida, mantenerla y posteriormente frenar. Mediremos las variables en el eje x de la simulación prestando especial interés al efecto que tendrán las diferentes pistas en las aceleraciones de cada robot puesto que esperamos que se cumpla que cuanto mayor sea el valor de la fricción mayor será la aceleración y antes vencerá la fuerza de rozamiento.

	Pista 1	Pista 2	Pista 3	Pista 4
Color del gráfico y de la pista	Amarillo	Verde	Azul	Rojo
Fricción	1	0.4	0.1	0
Tiempo de llegada a velocidad máxima (s)	0.423	0.427	0.483	infinito
Posición inicial eje x (m)	0	0	0	0
Posición final eje x (m)	1.530	1.529	1.521	-0.002
Aceleración máxima (m/s^2)	± 7.985	± 2.748	± 0.832	0

Tabla 8.1: Resultados de la simulación de fricción. Color de la gráfica, valor de fricción de la pista, posiciones iniciales y finales y aceleraciones máximas y mínimas.

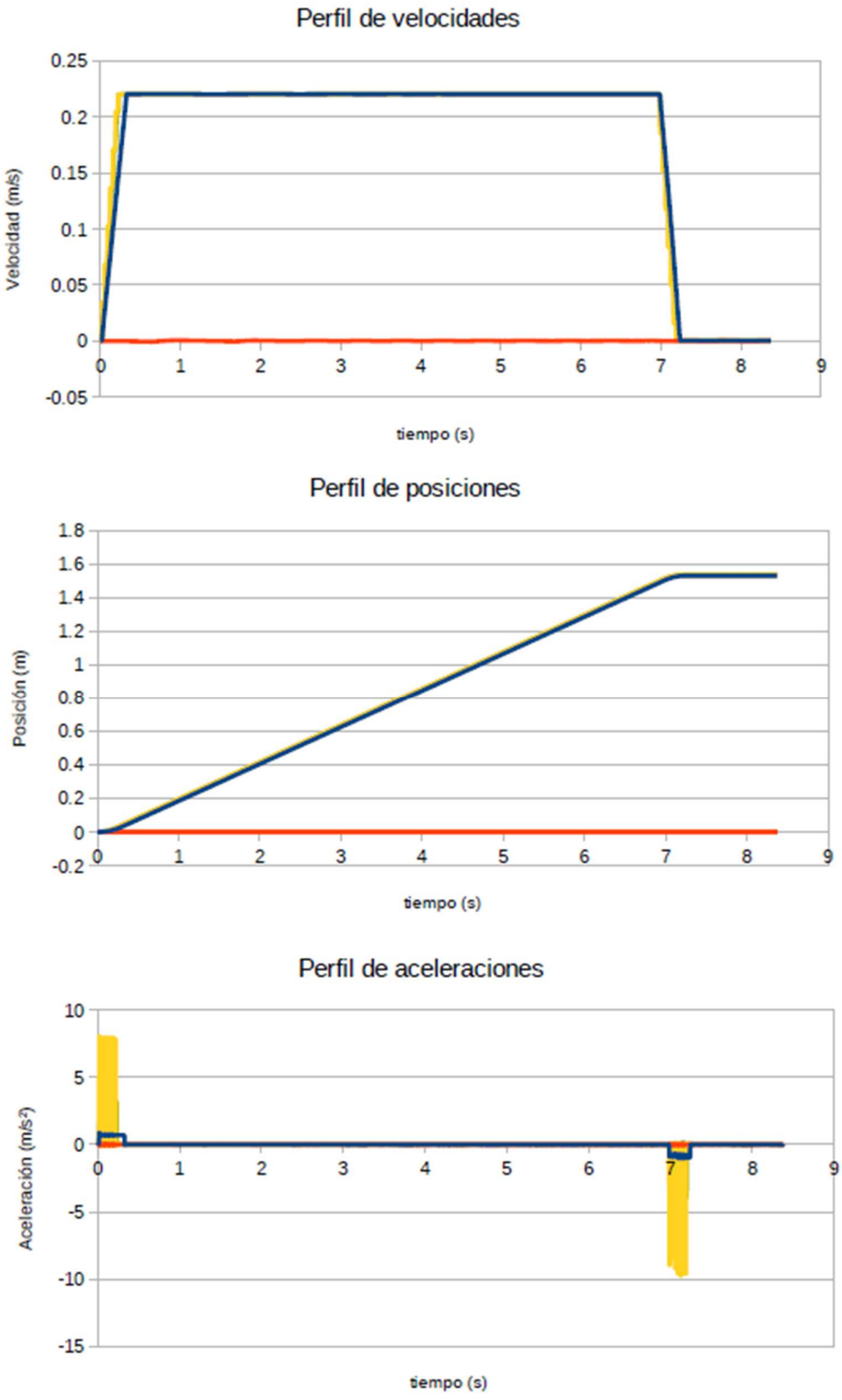

Figura 8.1: De arriba a abajo: Perfil de velocidad, perfil de posiciones y perfil de aceleraciones. En amarillo la gráfica corresponde al robot en la pista de fricción 1, en verde la gráfica corresponde al robot en la pista de valor 0.4, en azul la gráfica corresponde con el robot en la pista de fricción 0.1 y en rojo la gráfica que corresponde al robot en la pista de fricción 0.

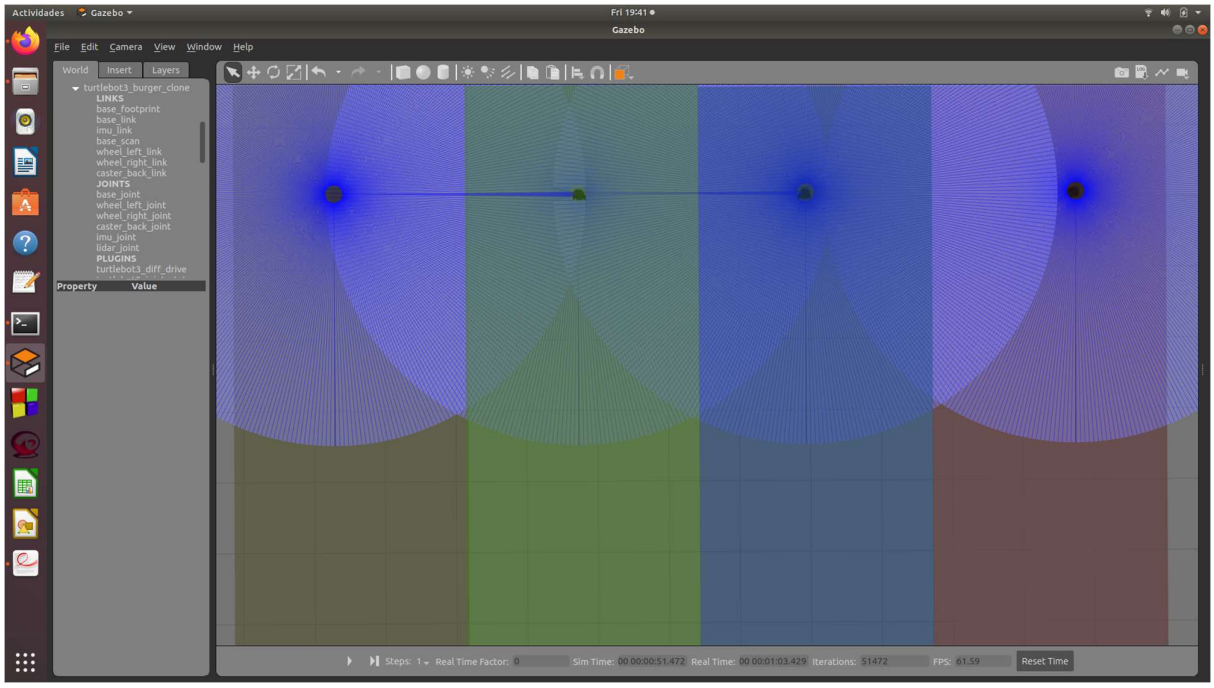

Figura 8.2: De izquierda a derecha: Pista amarilla fricción de valor 1, pista verde fricción de valor 0.4, pista azul fricción de valor 0.1 y pista roja fricción de valor 0.

En la *Figura 8.1* podemos comprobar cómo afecta el rozamiento al fijarnos en las aceleraciones que tomarán nuestros robots en cada pista, puesto que, para vencer la fuerza de rozamiento ante una pista con una rugosidad alta, la aceleración será mayor según la fórmula matemática $F = ma$ de tal forma que dicha fuerza pueda vencer la fuerza de rozamiento $F \geq F_r$. Si además nos fijamos en el perfil de velocidad, vemos como los robots en las pistas con fricción alta, 1 y 0.4, alcanzan antes el valor de velocidad pedido por teclado, mientras que el robot en la pista de fricción baja, 0.1, tarda algo más en llegar a la velocidad solicitada debido a la baja fricción, y lo mismo sucede cuando solicitamos al robot que frene completamente.

Es interesante resaltar el efecto de una pista sin ningún tipo de fricción puesto que, al no haber rugosidad alguna, la rueda y el suelo no tienen puntos de fricción con lo que no podrá moverse el robot. En este caso, debido a que el robot hace un movimiento oscilatorio muy leve sobre sí mismo al estar parado, el robot se encuentra en una posición final distinta de 0.

8.2. Efecto del deslizamiento.

Como explicamos anteriormente un objeto que se mueve en contacto con una superficie mojada pierde adherencia. Esto significa que las características propias del rozamiento entre el objeto y la superficie se ven reducidas, con lo cual, el robot al acelerar tardará en llegar al punto deseado.

Para hacer el experimento hemos fijado una velocidad de movimiento para dos robots idénticos que deben trabajar ante dos superficies distintas, una deslizante y la otra no deslizante.

	Pista normal	Pista deslizante
Color del gráfico y de la pista	Azul	Rojo
Índice de deslizamiento (adimensional)	0	0.5
Velocidad fijada (m/s)	0.220	0.220
Tiempo en alcanzar la velocidad pedida (s)	0.207	4.385
Tiempo que tarda en frenar (s)	0.218	4.39
Posición para t = 4.3s (m)	0.800	0.615
Aceleración máxima y mínima (m/s^2)	± 8.154	± 0.293

Tabla 8.2: Tabla de resultados experimentales de la simulación del efecto del deslizamiento. Color correspondiente a la gráfica, parámetros índices de deslizamiento, parámetros velocidad fijada, tiempo que tardan en llegar a la velocidad fijada, tiempo que tardan en frenar, posición para t = 4.3s y aceleraciones máximas y mínimas

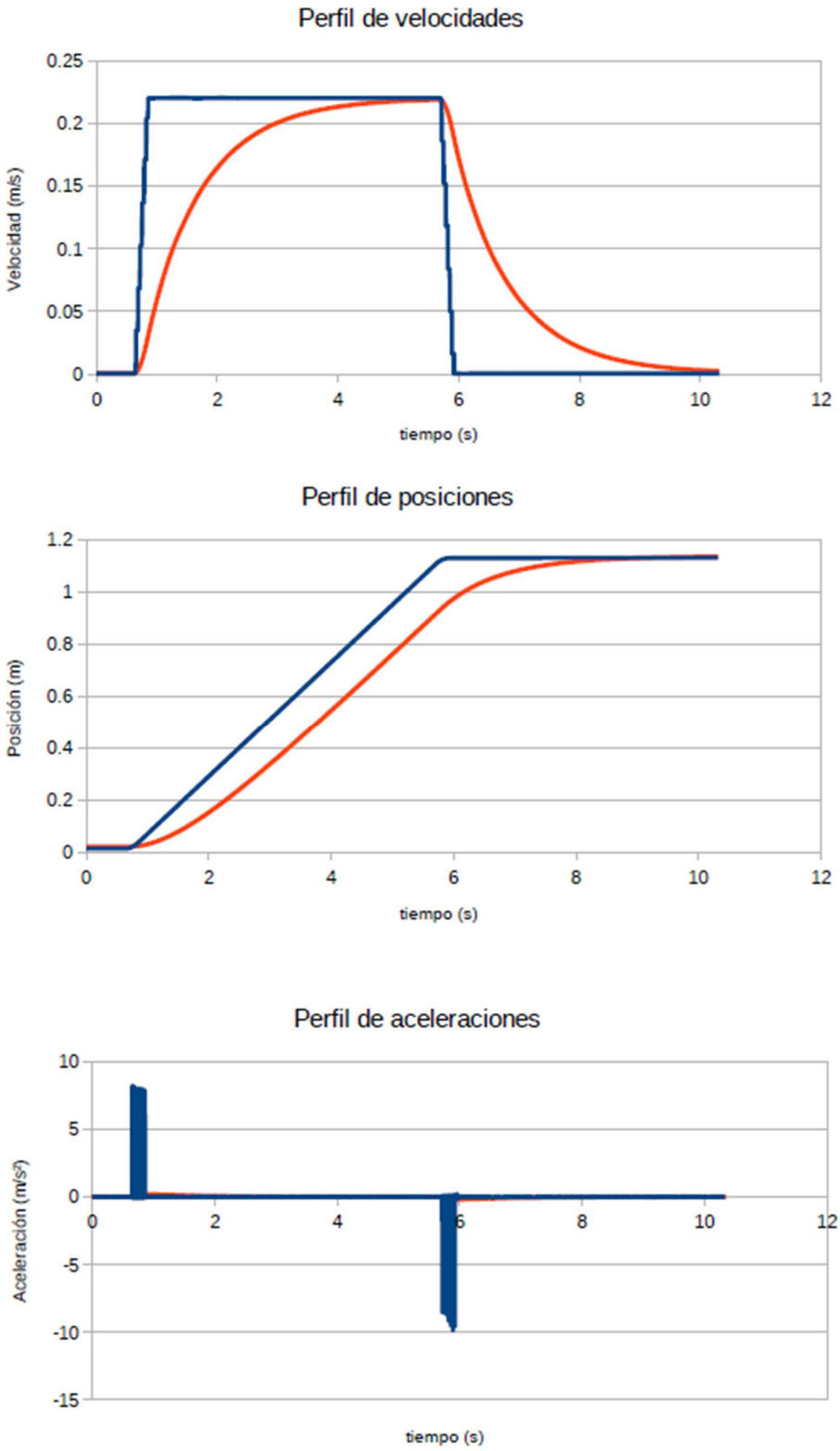

Figura 8.3: De arriba abajo: Perfil de velocidades, perfil de posiciones y perfil de aceleraciones. En azul la gráfica correspondiente a el robot en una pista si deslizamiento y en rojo el robot en una

pista con deslizamiento de 0.5. Debido al deslizamiento no alcanza la velocidad pedida hasta pasados 5 segundos y la posición por tanto está retardada con respecto a la pista sin deslizamiento.

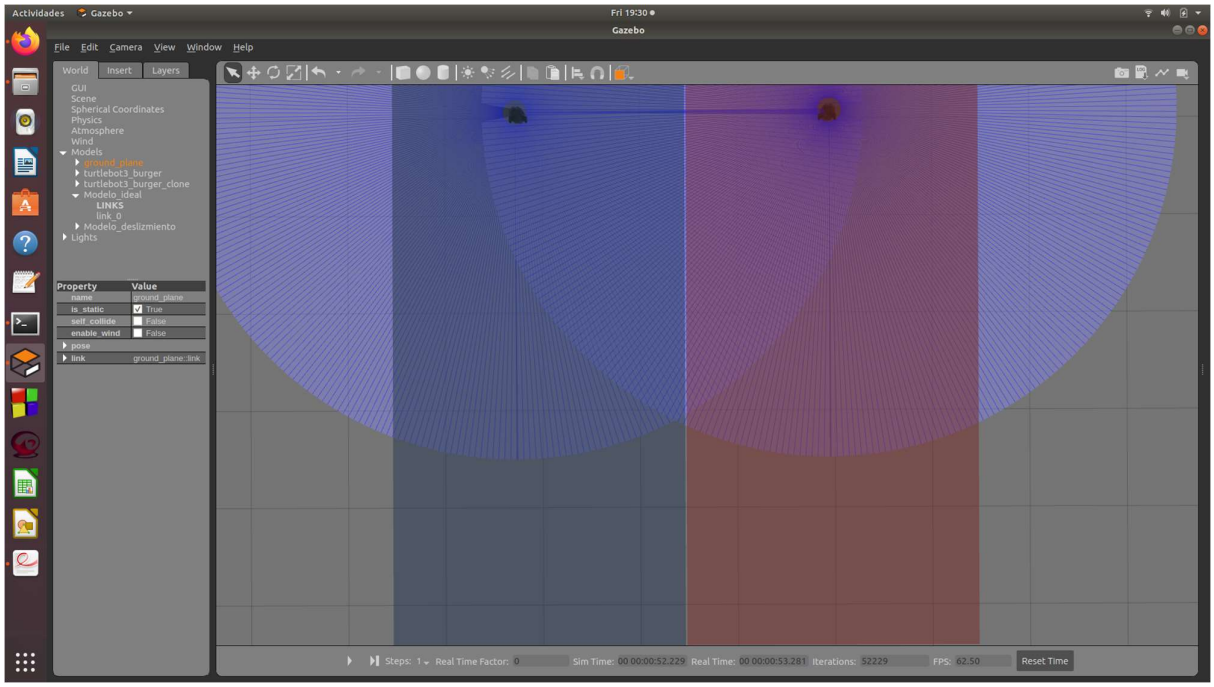

Figura 8.4: De izquierda a derecha: Pista azul sin deslizamiento y pista roja con valor de deslizamiento de 0.5.

Como podemos comprobar a la vista de la *Figura 8.3*, el robot que circula en la pista deslizante tardará alrededor de 5 segundos en alcanzar la velocidad que hemos solicitado por teclado. Esto hace que la posición con respecto al robot que circula en la pista sin deslizamiento se vea claramente afectada, produciéndose un desfase de 0.2 m de posición y algo más de 1 s de tiempo. Además, al pedir que frenen completamente, el robot que trabaja en el suelo deslizante tarda al menos 4 segundos en alcanzar velocidad nula debido a que la inercia del movimiento del robot en dicha pista lo mantiene moviéndose pese a que se haya dejado de mover las ruedas deslizándose por la pista hasta llegar a frenar.

8.3. Efecto de la masa del robot.

El objetivo de la simulación actual es entender el efecto de la masa del robot en su movimiento. Para ello hemos diseñado a partir del robot de estudio original otros tres robots cambiando sus parámetros de masas del script de descripción del robot.

Para cuantificar el efecto de las diferentes masas vamos a estudiar la posición en el espacio que ocuparán los distintos robots ante una misma orden de movimiento, con lo cual la posición será nuestra variable de estudio que servirá para cuantificar el efecto.

La simulación por tanto consta de cuatro robots con diferentes masas y se moverán usando un algoritmo de navegación básico basado en introducir unas coordenadas, en este caso tan solo le solicitamos que avance 2m en el eje de las x, por tanto, los cuatro robots empiezan a moverse para alcanzar el objetivo que se les solicita.

Color en la gráfica	Azul	Rojo	Amarillo	Verde
Peso del robot (kg)	1	5	10	50
Posición inicial (m)	0	0	0	0
Posición final (m)	1.21	1.18	1.16	1.06

Tabla 8.3: Correspondencia de la gráfica, masas de los distintos robots, posiciones iniciales y finales.

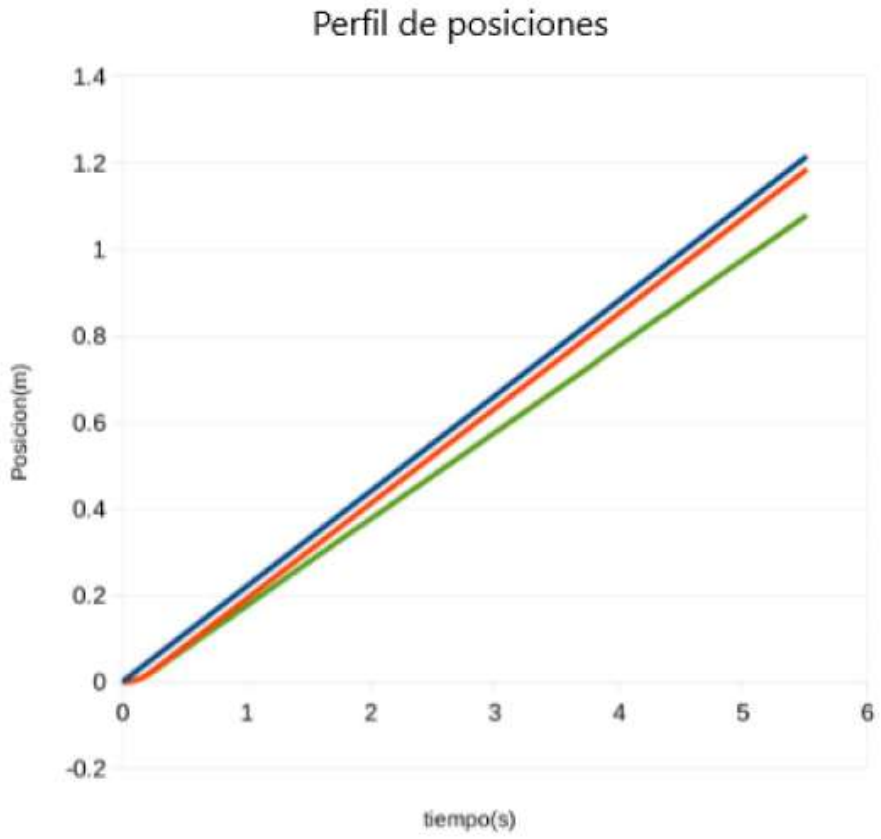

Figura 8.5: Posición de robots con distinto peso a lo largo del tiempo. De mayor a menor posición final: el robot de 1 kg, recta de color azul, llega a un desplazamiento final de 1.21 m, el robot de 5 kg, recta de color rojo, llega a un desplazamiento final de 1.18 m, el robot de 10kg, recta de color amarillo, llega a un desplazamiento final de 1.16 m y el robot de 50 kg, llega a un desplazamiento final de 1.06 m.

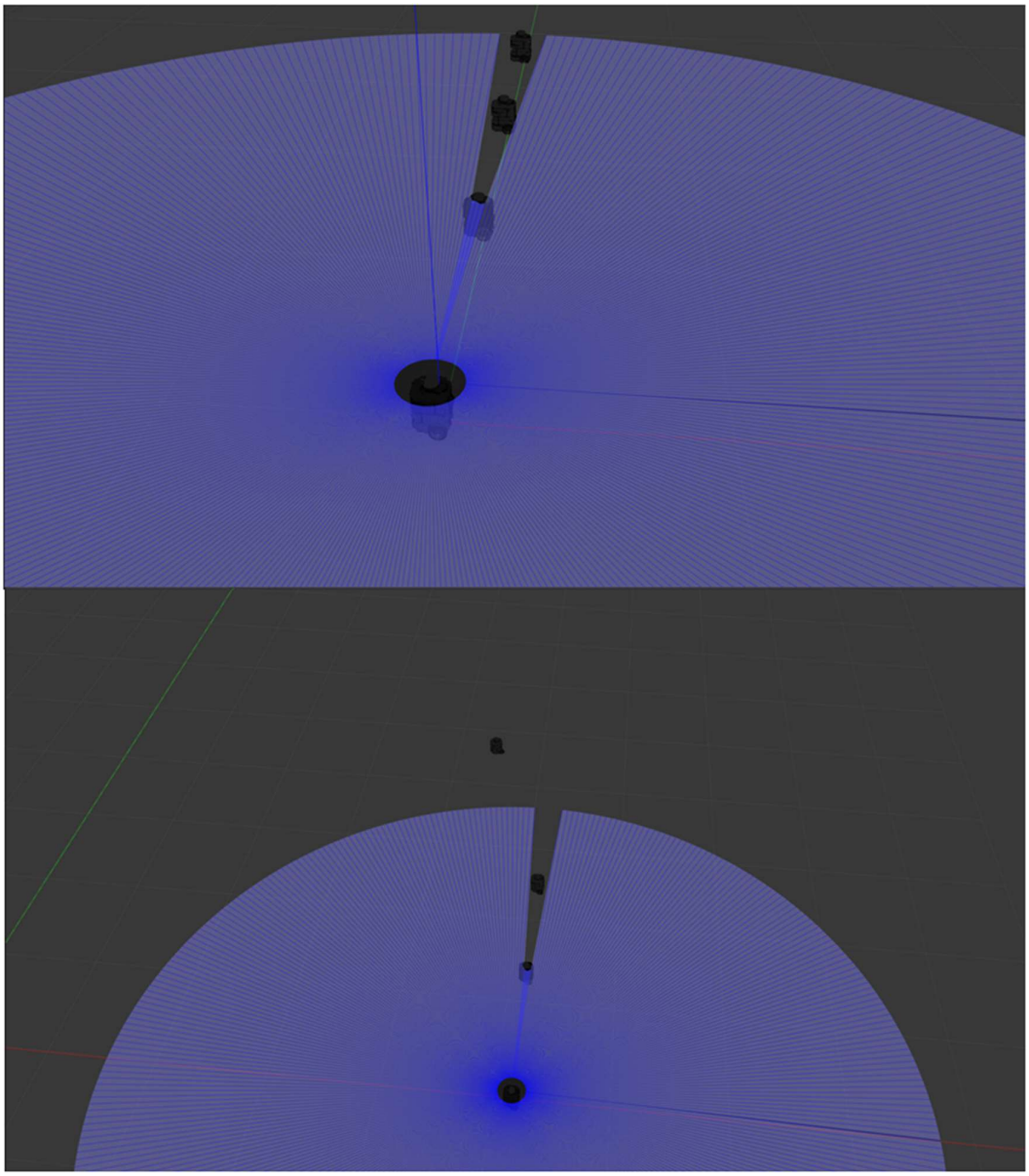

Figura 8.6: De arriba a abajo: en posición inicial robot de 50kg, robot de 10kg, robot de 5kg y robot de 1kg. En movimiento robot de 50kg se desvía por ser demasiado pesado, robot de 10kg, robot de 5kg y robot de 1kg.

Los robots que se encuentran dentro del rango de peso con el que puede trabajar el robot (15kg máximo) mantienen una posición aceptablemente cercana a la posición del robot de menor peso. En cambio, una vez superamos el umbral máximo de 15kg, nos encontramos con que el robot va a alejarse cada vez más de la posición que tiene el robot de 1kg y además debido a que el peso es

superior al que puede soportar el robot, éste se desequilibra y su orientación cambia dejando de circular exclusivamente en el eje x.

8.4. Efecto de un terreno no lineal en el robot.

Para estimar el efecto de un terreno no lineal en nuestro robot hemos diseñado tres superficies con inclinaciones distintas por las que va a intentar moverse.

Las superficies de trabajo serán con inclinación 5º, 10º y 25º. Para introducir estos valores en nuestra simulación tenemos que transformar los valores de grados a radianes.

En esta primera simulación nos fijaremos en el efecto de las rampas en la posición y la aceleración fijando una velocidad de movimiento para cada robot.

Color en la gráfica	Azul	Rojo	Amarillo
Inclinación en grados (º)	5	10	25
Inclinación en radianes (rad)	$\frac{1}{18}\pi$	$\frac{1}{9}\pi$	$\frac{5}{18}\pi$
Velocidad fijada (m/s)	0.22	0.22	0.22
Posición inicial (m)	0	0	0
Posición final (m)	2.3	2.1	1.4

Tabla 8.4: Conversión de la inclinación de grados a radianes, correspondencia de la gráfica y posiciones finales.

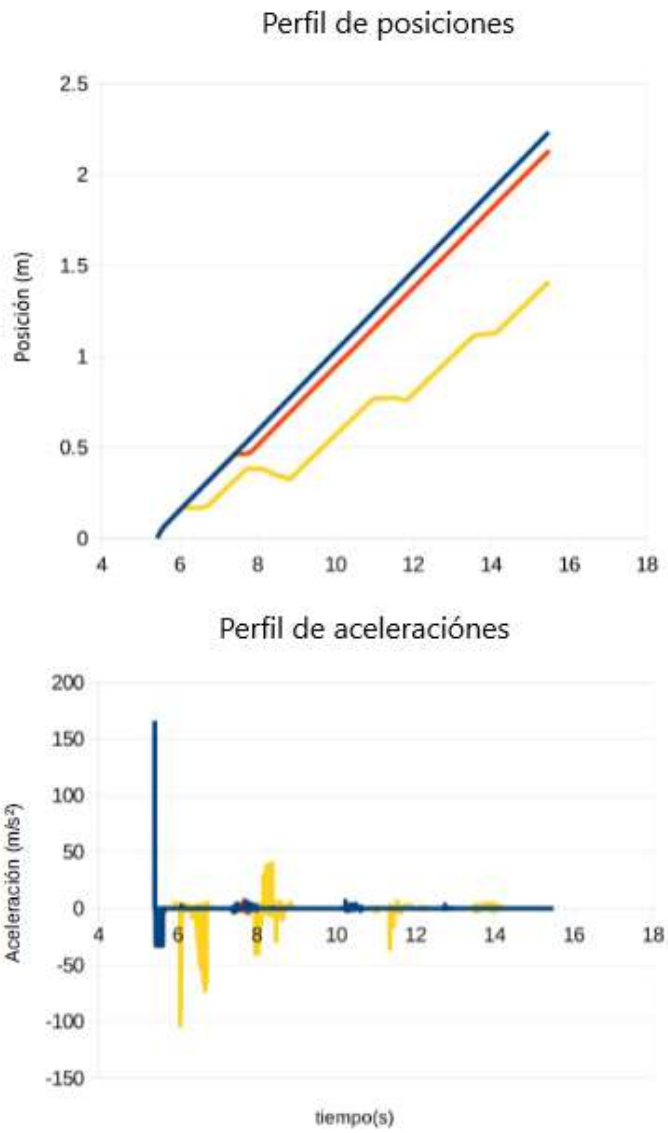

Figura 8.7: De arriba abajo: Perfil de posiciones y perfil de aceleraciones. De izquierda a derecha: la gráfica de color azul se corresponde con el robot que trabaja en una superficie de 5º de inclinación, la gráfica de color amarillo se corresponde con el robot que avanza en la rampa de 10º de inclinación y la gráfica de color verde se corresponde con el robot que trabaja en una superficie de 25º. Las aceleraciones actúan intentando compensar y estabilizar al robot ante las alteraciones producidas por las inclinaciones que desestabilizan al robot cuanto mayor es la inclinación.

La *Figura 8.7* nos muestra el efecto que tiene el contacto con la inclinación del terreno en la posición del robot y en la gráfica inferior vemos cómo las aceleraciones de cada robot intentan compensar esa inclinación para fijar una velocidad.

Resulta de especial interés ver cómo el robot en la rampa de 25º ve alterada su posición periódicamente al entrar en contacto con la rampa debido a que el robot pierde el equilibrio y empieza a oscilar con lo que al fijarnos en el efecto en su aceleración vemos como intenta compensar el efecto de la oscilación tras haber corregido, en parte, el efecto de la inclinación.

La siguiente simulación se hará con 4 robots tipo Burger que circularán en línea recta (eje x) por distintas pistas con inclinaciones que varían desde 0º hasta 25º. Para realizar la simulación, fijaremos una velocidad de movimiento para los 4 robots y comprobaremos como afecta en los perfiles de posición, velocidad y aceleración, de cada robot, la inclinación de la pista que le corresponda.

Color de la gráfica	Rojo	Azul	Amarillo	Verde
Inclinación Grados (º)	0	5	10	25
Inclinación Radianes (rad)	0	$\frac{1}{18}\pi$	$\frac{1}{9}\pi$	$\frac{5}{18}\pi$
Velocidad fijada (m/s)	0.22	0.22	0.22	0.22
Posición inicial (m)	0	0	0	0
Posiciones finales (m)	2.17	2.12	2.03	1.44

Tabla 8.5: Conversión a radianes y robot que circula en cada rampa.

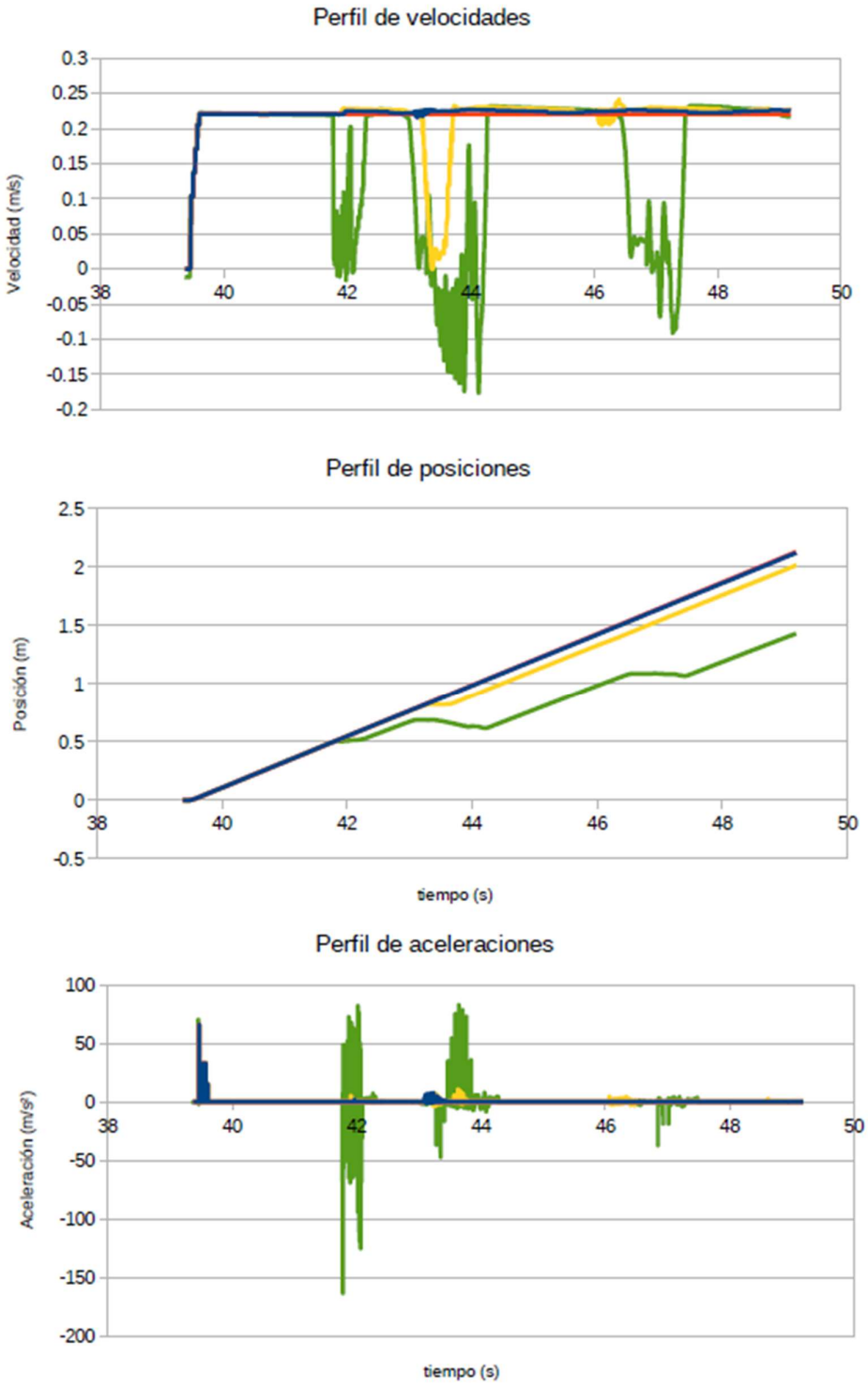

Figura 8.8: De arriba abajo: Perfil de velocidades, perfil de posiciones y perfil de aceleraciones. De izquierda a derecha: la gráfica de color rojo se corresponde al robot trabajando en una pista sin inclinación, la gráfica de color azul se corresponde con el robot trabajando en una pista de inclinación 5º, la gráfica de color amarillo se corresponde con el robot trabajando en una pista de

inclinación 10º y la gráfica de color verde se corresponde con el robot trabajando en una pista de inclinación 20º.

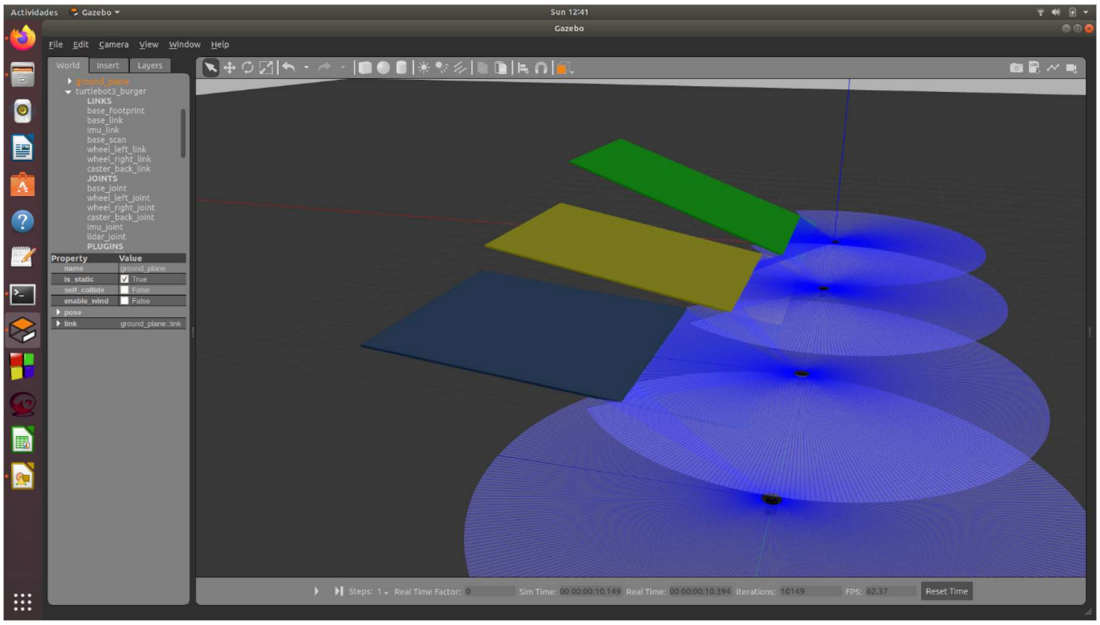

Figura 8.9: Simulación de terrenos no lineales. De izquierda a derecha: Robot en pista sin inclinación, robot en pista azul con inclinación 5º, robot en pista amarilla con inclinación 10º y robot en pista verde con inclinación 20º.

Como podemos observar en la *Figura 8.8* la posición del robot que trabaja en una superficie sin inclinación (0º) se mueve de forma lineal sin efecto alguno en su movimiento. Cuando nos fijamos en el robot que va a moverse en una superficie de 5º vemos como su posición al entrar en contacto con la superficie de 5º varía, pero no de forma especialmente significativa. En cambio, el robot que avanza en la rampa de 10º al alcanzarla se ve cómo su posición se retrasa significativamente con respecto a los robots anteriores. Además, podemos ver cómo su velocidad varía y hay un tiempo de un segundo en el que la curva de velocidad cae a 0 hasta que el sistema de control reacciona para intentar llevar su velocidad a la que habíamos pedido por la consola de comandos. Éste mismo error se ve acentuado significativamente en el caso del robot que circula en un plano de 25º puesto que la inclinación es tan grande que al entrar en contacto con la rampa el robot se desequilibra y pierde velocidad y la posición se ve afectada significativamente con respecto a las anteriores puesto que, aunque está poco a poco avanzando por la rampa, las continuas oscilaciones del robot hacen que su posición se vea especialmente afectada.

8.5.- Efecto de cambios en el código de navegación.

Para comprobar los efectos de los cambios en los parámetros del algoritmo de navegación hemos realizado tres simulaciones con distintos parámetros en cada una.

8.5.1. Tabla de relaciones de comportamiento.

		Simulación 1	Simulación 2	Simulación 3 (*)
Posición inicial (m)	x	- 2	- 2	- 2
	y	- 0.5	- 0.5	- 0.5
Posición objetivo (m)	x	2	2	2
	y	0.5	0.5	0.5
Posición final (m)	x	1.76	1.48	0.23
	y	0.46	- 0.37	- 0.8
Tiempo hasta llegar al objetivo (s)		19.874	18.599	Error

Tabla 8.6: Parámetros de análisis comparativo entre las simulaciones. Posiciones iniciales, objetivo, finales y tiempo de simulación hasta alcanzar el objetivo

8.5.2.- Simulación 1: sin cambios en el código.

A continuación, veremos cómo es la simulación bajo los parámetros de comportamiento que vienen por defecto en el algoritmo de navegación. Llamaremos al programa cargando el mapa de simulación y finalmente le ordenamos al robot que se desplace a un punto fijado en Rviz2.

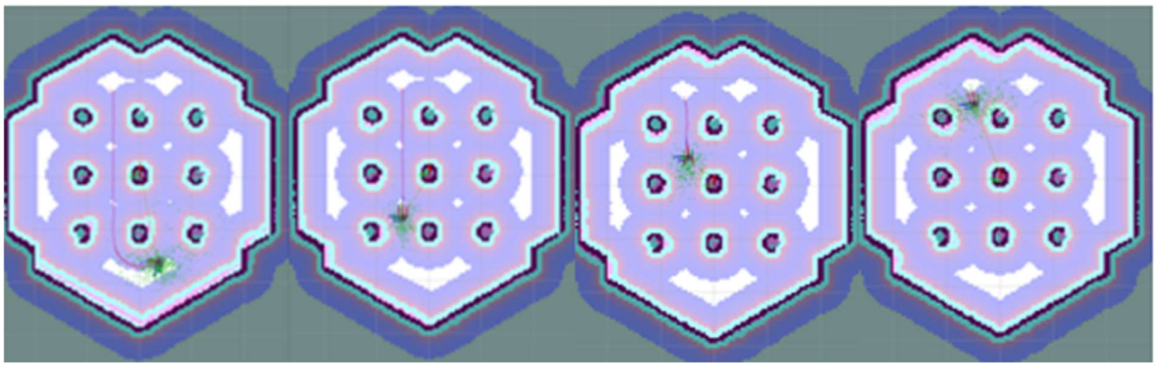

Figura 8.10: Comportamiento del algoritmo sin modificar los parámetros. De izquierda a derecha: Le enviamos la orden de posición final al robot y crea la ruta más corta en el mapa, el robot comienza su movimiento siguiendo la ruta establecida, el robot se acerca a la posición que hemos pedido por teclado y finalmente alcanza la posición que buscábamos.

Podemos ver que el algoritmo detecta en el mapa el camino más corto para llegar al objetivo sorteando los obstáculos presentes en el mapa sin modificarse a lo largo del trayecto.

8.5.3.- Simulación 2: Cambios en los parámetros sim_time, planificador y meta.

Para realizar la simulación vamos a cambiar algunos parámetros internos del código. Vamos a modificar el valor del sim_time para que a la capacidad de reacción del robot a girar sea limitada, el valor de la tolerancia a la que el robot detectará haber llegado a la meta, el valor al que el robot genera la planificación de rutas intentando mantener una distancia de los obstáculos y el valor por el cual puede acercarse a los obstáculos. Después, llamamos al programa y fijamos un punto de destino en la ventana de RVIZ2 cargada para que el robot se mueva al objetivo fijado en un mapa donde se encuentran varios obstáculos fijos que se ha cargado previamente en la memoria del robot.

```
dwb_controller:
  ros__parameters:
    use_sim_time: False
    debug_trajectory_details: True
    min_vel_x: 0.0
    min_vel_y: 0.0
    max_vel_x: 0.22
    max_vel_y: 0.0
    max_vel_theta: 1.0
    min_speed_xy: 0.0
    max_speed_xy: 0.22
    min_speed_theta: 0.0
    min_x_velocity_threshold: 0.001
    min_y_velocity_threshold: 0.5
    min_theta_velocity_threshold: 0.001
    acc_lim_x: 2.5
    acc_lim_y: 0.0
    acc_lim_theta: 3.2
    decel_lim_x: -2.5
    decel_lim_y: 0.0
    decel_lim_theta: -3.2
    vx_samples: 20
    vy_samples: 5
    vtheta_samples: 20
    sim_time: 3.0 #1.7
    linear_granularity: 0.05
    xy_goal_tolerance: 1.0 #0.25
    transform_tolerance: 0.8 #0.2
    critics: ["RotateToGoal", "Oscillation", "BaseObstacle", "GoalAlign", "PathAlign", "PathDist",
"GoalDist"]
    BaseObstacle.scale: 0.02
    PathAlign.scale: 0.0
    GoalAlign.scale: 0.0
    PathDist.scale: 32.0
    GoalDist.scale: 24.0
    RotateToGoal.scale: 32.0

local_costmap:
  local_costmap:
    ros__parameters:
      use_sim_time: False
      global_frame: odom
      plugin_names: ["obstacle_layer", "inflation_layer"]
      plugin_types: ["nav2_costmap_2d::ObstacleLayer", "nav2_costmap_2d::InflationLayer"]
      rolling_window: true
      width: 3
      height: 3
      resolution: 0.05
      robot_radius: 0.105
      inflation_layer.inflation_radius: 1.0 #0.2
      inflation_layer.cost_scaling_factor: 2.0 #1.0
      obstacle_layer:
        enabled: True
      always_send_full_costmap: True
```

Figura 8.11: Fracción del código con cambios en parámetros de la simulación 1. Los parámetros originales en el código vienen precedidos de una almohadilla y se encuentran en color azul.

Al haber aumentado el valor de *sim_time* hacemos que el robot no pueda girar de forma brusca, la ruta se planificará 1m de distancia de los obstáculos y el sistema de navegación detectará haber llegado al punto objetivo a una distancia de 1m de distancia del punto fijado como objetivo.

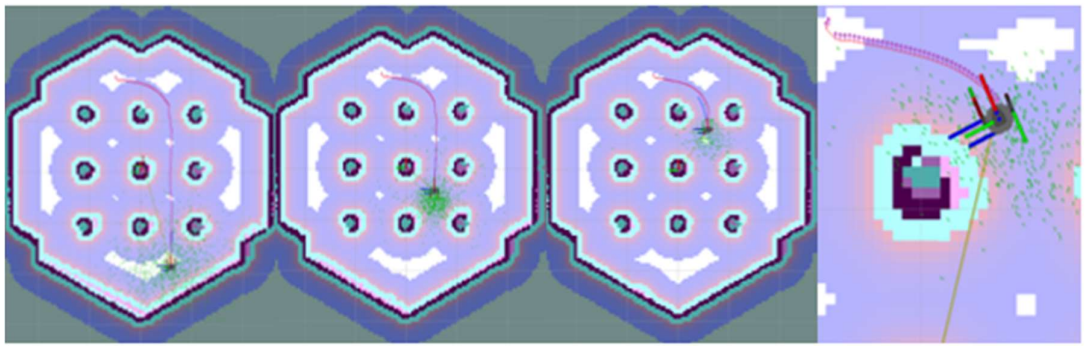

Figura 8.12: Funcionamiento del algoritmo tras alterar sus parámetros. De izquierda a derecha: Le enviamos la orden de posición final al robot y crea la ruta más corta en el mapa teniendo en cuenta que debe mantener una distancia de 1 m de los obstáculos y que no puede realizar giros bruscos, el robot comienza su movimiento siguiendo la ruta establecida, el robot se acerca a la posición que hemos pedido por teclado y finalmente alcanza la posición a una distancia de 1m del objetivo.

Como podemos observar en la *Figura 8.12*, tras enviar la posición deseada al robot, éste genera una ruta buscando que sea lo más corta posible, al comenzar el movimiento se encuentra con los obstáculos cargados en el mapa que sorteará sin problemas mientras mantiene la ruta fijada hasta llegar a alcanzar una posición alejada a 1m de distancia del objetivo.

8.5.4.- Simulación 3: Cambios en los parámetros sim_time, planificador, velocidades y meta.

Para realizar la simulación vamos a cambiar algunos parámetros internos del código. Vamos a modificar el valor del sim_time para que a la capacidad de reacción del robot a girar sea aún más limitada, la velocidad máxima a la que podrá moverse el robot durante la simulación, el valor de la tolerancia a la que el robot detectará haber llegado a la meta, el valor al que el robot genera la planificación de rutas intentando mantener una distancia de los obstáculos y el valor por el cual puede acercarse a los obstáculos. Después, llamamos al programa y fijamos un punto de destino en la ventana de RVIZ2 cargada para que el robot se mueva al objetivo fijado en un mapa donde se encuentran varios obstáculos fijos que se ha cargado previamente en la memoria del robot.

```
dwb_controller:
  ros__parameters:
    use_sim_time: False
    debug_trajectory_details: True
    min_vel_x: 0.0
    min_vel_y: 0.0
    max_vel_x: 2.2 #0.22
    max_vel_y: 0.0
    max_vel_theta: 1.0
    min_speed_xy: 0.0
    max_speed_xy: 2.2 #0.22
    min_speed_theta: 0.0
    min_x_velocity_threshold: 0.001
    min_y_velocity_threshold: 0.5
    min_theta_velocity_threshold: 0.001
    acc_lim_x: 2.5
    acc_lim_y: 0.0
    acc_lim_theta: 3.2
    decel_lim_x: -2.5
    decel_lim_y: 0.0
    decel_lim_theta: -3.2
    vx_samples: 20
    vy_samples: 5
    vtheta_samples: 20
    sim_time: 3.7 #1.7
    linear_granularity: 0.05
    xy_goal_tolerance: 0.50 #0.25
    transform_tolerance: 0.50 #0.2
    critics: ["RotateToGoal", "Oscillation", "BaseObstacle", "GoalAlign", "PathAlign", "PathDist",
"GoalDist"]
    BaseObstacle.scale: 0.02
    PathAlign.scale: 0.0
    GoalAlign.scale: 0.0
    PathDist.scale: 32.0
    GoalDist.scale: 24.0
    RotateToGoal.scale: 32.0

local_costmap:
  local_costmap:
    ros__parameters:
      use_sim_time: False
      global_frame: odom
      plugin_names: ["obstacle_layer", "inflation_layer"]
      plugin_types: ["nav2_costmap_2d::ObstacleLayer", "nav2_costmap_2d::InflationLayer"]
      rolling_window: true
      width: 3
      height: 3
      resolution: 0.05
      robot_radius: 0.105
      inflation_layer.inflation_radius: 1.0 #0.2
      inflation_layer.cost_scaling_factor: 2.0 #1.0
      obstacle_layer:
        enabled: True
      always_send_full_costmap: True
```

Figura 8.13: Fracción del código con cambios en los parámetros de la simulación 2. Los parámetros originales en el código vienen precedidos de una almohadilla y se encuentran en color azul.

Los cambios realizados en la tercera simulación hacen que el sistema de navegación falle. Hemos fijado un sim_time aún mayor, una velocidad de trabajo significativamente superior, bajamos la tolerancia a la meta y mantenemos los valores de generación de la trayectoria y cercanía a los obstáculos.

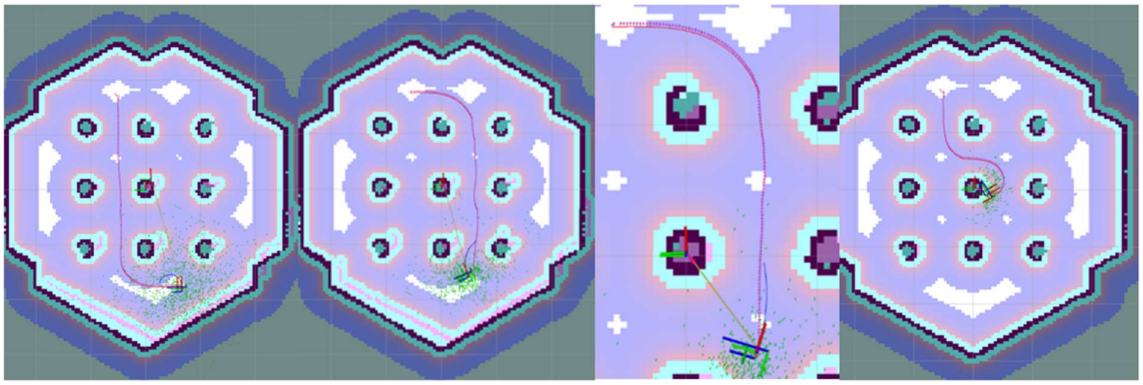

Figura 8.14: Alteraciones en el algoritmo de navegación por la modificación de sus parámetros. De izquierda a derecha: Le enviamos la orden de posición final al robot y crea la ruta más corta en el mapa, el robot comienza su movimiento y crea una nueva ruta debido a que inicia el movimiento a una velocidad demasiado alta, el robot se desplaza siguiendo el nuevo camino pero debido a su alta velocidad y su baja capacidad de giro tiene que frenar bruscamente para evitar colisionar con la columna que se encuentra a su izquierda y al hacerlo vuelve a crear una nueva ruta.

Al haber aumentado el valor de sim_time hacemos que el robot no pueda girar de forma brusca y además la velocidad que alcanza el robot es significativamente superior a la de las simulaciones anteriores, con lo cual, al intentar generar el camino hasta el punto objetivo, el sistema de navegación tendrá que cambiar el camino a uno menos brusco. Además, le hemos subido el valor de detección de obstáculos y de esquiva de los mismos al robot, por lo que, cuando tiene que pasar por las columnas al intentar estar a 1m de distancia de cada obstáculo vuelve a generar una nueva trayectoria. A partir de ese momento, el robot frena bruscamente para evitar colisionar con el obstáculo central. Tras haber frenado para evitar la colisión, el algoritmo sigue funcionando e intenta redireccionar el robot, pero al no poder girar bruscamente y tener una velocidad muy alta, no le da tiempo a poder reposicionarse y colisiona contra la columna que está posicionada a la derecha de la columna central.

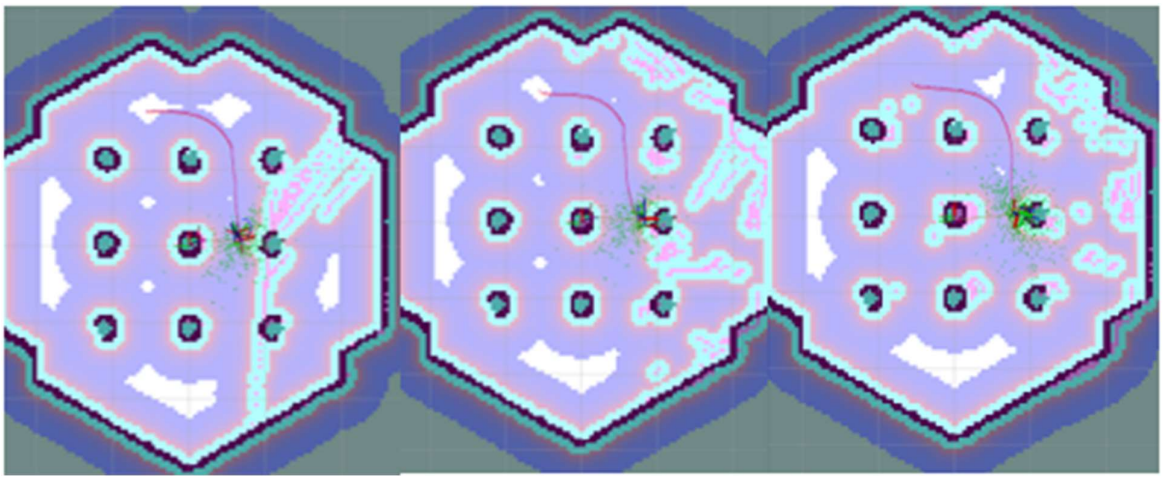

Figura 8.15: Secuencia de colisión del robot. De izquierda a derecha: El robot intenta retomar la nueva ruta, pero la velocidad que alcanza es demasiado alta y no le da tiempo a girar para evitar el obstáculo que supone la columna y colisiona contra ella.

Como podemos ver en la *Figura 8.15* los cambios realizados en los parámetros internos del sistema de navegación han hecho que éste falle debido a que los cambios realizados en el código son extremos. La alta velocidad, la falta de capacidad de reacción de giro y la capacidad a acercarse a obstáculos, no permiten que el algoritmo se ejecute correctamente.

9. Análisis y conclusiones.

Todo lo que hemos estudiado es, en definitiva, los efectos de las aproximaciones en simulación a los efectos de la realidad en cuanto al trabajo que puede ejecutar un robot. Donde encontramos que el propio entorno puede resultar variable y por ello puede generar problemas en un sistema programado para un entorno y una acción concreta. Para probar las simulaciones en un robot real, habría que ajustar las características propias del robot y de los entornos no lineales que busquemos probar. Esto significa que deberíamos medir la fricción entre el robot y el suelo donde vayamos a trabajar. Además, habría que comprobar las irregularidades que pueda tener el suelo y la inclinación del mismo.

Hemos probado los efectos que tienen los efectos dinámicos sobre el algoritmo de navegación y sobre el rendimiento del propio robot. Las condiciones de trabajo que hemos impuesto al robot son en algunos casos extremas o incluso condiciones que no son propias del robot con el que trabajamos, como puede ser haber simulado el movimiento del robot con un peso extremadamente superior al máximo peso con el que podría trabajar el robot de trabajo.

He puesto en práctica conocimientos propios desarrollados a lo largo de la carrera. Para generar las simulaciones y extraer la información necesaria para parametrizar, he tenido que pensar en qué medir, de las posibles variables, para extraer la información y qué variables no aportarían información. Además, para poder realizar las gráficas y comprobar los datos extraídos, dado que el trabajo se ha realizado en un sistema operativo en base Linux, he aprendido a trabajar con hojas de cálculo de OpenOffice.

Durante el desarrollo del trabajo, ha habido varios aprendizajes relacionados con conceptos no estudiados a lo largo de la carrera. He tenido que aprender a trabajar con un sistema operativo en base Linux, trabajar con lenguajes de programación que no se han desarrollado en demasía a lo largo de la carrera como pueden ser Python o C++. He aprendido a trabajar con librerías publicas desde git, comprendido el funcionamiento y cómo programar con ROS2.

10. Anexos.

10.1 Anexo I.

En el anexo I encontramos tres carpetas en las cuales podemos encontrar los códigos que han sido usados a lo largo del trabajo para hacer las simulaciones. Las tres carpetas son:

- My_package_simulations
  - Encontraremos dos carpetas en las cuales se encuentra el entorno de simulación en gazebo y el archivo de extensión .py que ejecuta la acción de lanzar con ros2.

- Navigation2_parámetros
  - Encontraremos los parámetros que usa el paquete para cargar y ejecutar las características de Navigation2 para la simulación y los mapas que se pueden cargar. El mapa usado en la simulación es map.yaml aunque se pueden encontrar otros mapas en la carpeta correspondiente, de diseño propio, que se han usado para probar el funcionamiento del Navigation2.

- Scripts movimiento
  - Encontraremos un script de movimiento automático en c++, un script de movimiento ejecutado por teclado en Python y un script de movimiento por el cual le solicitamos al robot que se desplace una cantidad de metros y que pueda girar sobre sí mismo.

11. Bibliografía.

[1] K. C. Translated, P. Selver, and N. Playfair, "Rossum's Universal Robots."

[2] L. Yukou and L. Yukou, *Lie Zi. El libro de la perfecta vacuidad*. Traducción directa del chino, introducción y notas a cargo de Iñaki Preciado Idoeta. Barcelona: Editorial Kairós, 1987.

[3] R. M. plc. and R. M. plc., *The Hutchinson Dictionary of Scientific Biography*. Abingdon, Oxon: Helicon Publishing, 2004.

[4] Homero, *Iliada,150*, vol. Clasicos G. Madrid, 1996.

[5] s. I. a. C. Homero, C. García Gual, and J. Flaxman, *Odisea*. Alianza, 2004.

[6] D. R. Hill, *A history of engineering in classical and medieval times*. La Salle, Ill.: Open Court Pub.

[7] M. Taddei, *I robot di Leonardo da Vinci : la meccanica e i nuovi automi nei codici svelati*. Leonardo3, 2007.

[8] "Autómatas en la Historia. Jacques de Vaucanson | Actually Notes Magazine," *Actually Notes Mag.*, 2016, Accessed: May 19, 2020. [Online]. Available: https://www.actuallynotes.com/automatas-en-la-historia-jacques-de-vaucanson-html/.

[9] "Karakuri - Wikipedia, la enciclopedia libre." Disponible en: https://es.wikipedia.org/wiki/Karakuri (accessed May 19, 2020).

[10] *Designing Tomorrow: America's World's Fairs of the 1930s Exhibit Preview*. The Henry Ford Museum, 2013.

[11] "Elektro - Wikipedia." Disponible en: https://en.wikipedia.org/wiki/Elektro (accessed May 19, 2020).

[12] I. Asimov, *Los robots*. Barcelona: Ediciones Martínez Roca, 1984.

[13] "George Devol - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/George_Devol (accessed May 19, 2020).

[14] J. F. Engelberger, *Robotics in service*. MIT Press, 1989.

[15] S. Y. Nof, *Handbook of industrial robotics*, 2ª edición. John Wiley, 1999.

[16] A. Witze, "Space rovers in record race," *Nature*, vol. 498, no. 7454. pp. 284–285, Jun. 19, 2013, doi: 10.1038/498284a.

[17] "Rover (space exploration) - Wikipedia." Disponible en: https://en.wikipedia.org/wiki/Rover_(space_exploration)#Prop-M (accessed May 19, 2020).

[18] "La historia de KUKA: Automatización en el pasado y en el presente | KUKA AG."

Disponible en: https://www.kuka.com/es-es/acerca-de-kuka/historia (accessed May 19, 2020).

[19] "Victor Scheinman - Wikipedia, la enciclopedia libre." Disponible en: https://es.wikipedia.org/wiki/Victor_Scheinman (accessed May 19, 2020).

[20] "Viking 1 - Wikipedia, la enciclopedia libre." Disponible en: https://es.wikipedia.org/wiki/Viking_1 (accessed May 19, 2020).

[21] "ROS.org | History." Disponible en: https://www.ros.org/history/ (accessed May 19, 2020).

[22] "Why ROS 2?" Disponible en: https://design.ros2.org/articles/why_ros2.html (accessed Aug. 24, 2020).

[23] "Física para la ciencia y la tecnología. I - Paul Allen Tipler, Gene Mosca - Google Libros." Disponible en: https://books.google.es/books?id=9MFLer5mAtMC&pg=PA319&dq=masa+inercial&hl=es&sa=X&ved=2ahUKEwil6c7uk7TrAhXLBGMBHdgmAhwQ6AEwAHoECAAQAg#v=onepage&q=masa inercial&f=false (accessed Aug. 24, 2020).

[24] "Masa inercial - Wikipedia, la enciclopedia libre." Disponible en: https://es.wikipedia.org/wiki/Masa_inercial (accessed Aug. 24, 2020).

[25] "Momento de inercia - Wikipedia, la enciclopedia libre." Disponible en: https://es.wikipedia.org/wiki/Momento_de_inercia (accessed Aug. 24, 2020).

[26] F. C. Park and K. M. Lynch, "Introduction to robotics_Motion Planning_Control."

[27] I. Díaz, "Mecánica de suelos -Lambe y Whitman." Accessed: Aug. 24, 2020. [Online]. Disponible en: https://www.academia.edu/37808584/Mecánica_de_suelos_Lambe_y_Whitman.

[28] "Fricción - Wikipedia, la enciclopedia libre." Disponible en: https://es.wikipedia.org/wiki/Fricción (accessed Aug. 24, 2020).

[29] "Gazebo : Tutorial : Friction." Disponible en: http://gazebosim.org/tutorials?tut=friction&cat=physics (accessed Aug. 24, 2020).

[30] "TurtleBot3." Disponible en: https://emanual.robotis.com/docs/en/platform/turtlebot3/ros2_setup/#hardware-setup (accessed Aug. 24, 2020).

[31] "Navigation Concepts — Navigation 2 1.0.0 documentation." Disponible en: https://navigation.ros.org/concepts/index.html (accessed Aug. 24, 2020).

[32] "TurtleBot3." Disponible en: https://emanual.robotis.com/docs/en/platform/turtlebot3/ros2_navigation2/#send-navigation-goal (accessed Aug. 24, 2020).

[33] P. Waurzyniak and P. Waurzyniak, "Masters of Manufacturing: Joseph F. Engelberger," *Soc. Manuf. Eng.*, vol. 137, no. 1, 2006, Accessed: May 19, 2020. [Online]. Disponible en: http://www.sme.org/cgi-bin/find-articles.pl?&ME06ART39&ME&20060709#article.

[34] S. Bermejo Sánchez, *Desarrollo de robots basados en el comportamiento*. Edicions UPC, 2003.

[35] "Robótica - Wikipedia, la enciclopedia libre." Disponible en: https://es.wikipedia.org/wiki/Robótica (accessed May 19, 2020).

[36] *Company History*. Fuji Yusoki Kogyo Co.

[37] *Industry Spotlight: Robotics from Monster Career Advice*. .

[38] "Poliarticulados - ROBOTICA," *sites.google.com*, Accessed: May 19, 2020. [Online]. Disponible en: https://sites.google.com/a/unitecnica.net/dmhenao/articulos/poliarticulados.

[39] *Definición de robótica - roboticspot.com*. .

[40] *Definición de robótica - RAE*. .

[1], [26], [33]–[40]